

ATARI
ST COMPUTER
Die Fachzeitschrift für den ATARI-ST Anwender.
Februar 86
ÖS 51,—/Sfr. 6,—
DM 6,—
NR. 2
* Monitor SM 124, scharfgemacht
* Dateiverwaltung
* Einführung in Pascal
* Druckanpassung
* Softwaretests
SEKA-Assembler
ST-Pascal
GEM
Das Fenster zur Zukunft



Liebe ST Leser!

Daß ein Computer, also auch der ATARI ST, ohne Software ziemlich nutzlos ist, sollte bekannt sein. Diese wichtige Tatsache hat auch ATARI erkannt und bietet nun nach dem CP/M Emulator schon wieder einige sogenannte **Public Domain** Programme an. Dies sind Programme, die **kostenlos** auf die eigene Diskette kopiert und weitergegeben werden können. Zu diesen neuen Programmen gehören ein Textverarbeitungsprogramm, eine Datenbank, ein Zeichen- und Malprogramm sowie eine Utility Diskette. Nähere Einzelheiten zu den Programmen sind in diesem Heft zu finden. ATARI gibt auf diesem Weg den ST Besitzern bzw. Käufern ein breites Spektrum an guter Softeware mit nach Hause. Ich will aber nicht verschweigen, daß die oben genannten Programme als Ersatz für GEM-Paint und GEM-Write erscheinen, die nicht mehr geliefert werden sollen. Allerdings bin ich der Meinung, daß dies für den Anwender ein sehr guter Tausch ist.

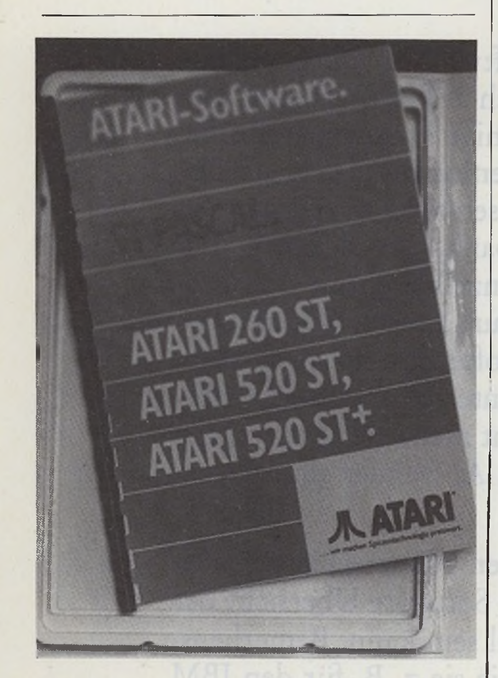
Aber nicht nur ATARI, sondern auch die kommerziellen Softwarehäuser bringen ständig neue Programme auf den Markt, so daß die Software-Übersicht der letzten Ausgabe in manchen Punkten nicht mehr aktuell sein kann. Bemerkenswert ist dabei, das professionelle Anwendersoftware, wie sie z. B. für den IBM PC existiert, schon jetzt, und teilweise in verbesserter Form, für den ATARI ST zu haben ist. Damit sind dem ATARI ST, der dem IBM PC in Bezug auf Rechenleistung und Speicherkapazität überlegen ist, alle Voraussetzungen gegeben, in der „Bürowelt" oder auch in Betrieben Fuß zu fassen.

In diesem Sinne ist die ST Redaktion eifrig bemüht, für Sie immer die neuste Software zu testen und ausführlich darüber zu berichten. Bleibt nur noch zu hoffen, daß das rasante Wachsen des Softwareangebotes noch eine Weile andauert.

Inhalt

Allgemeines

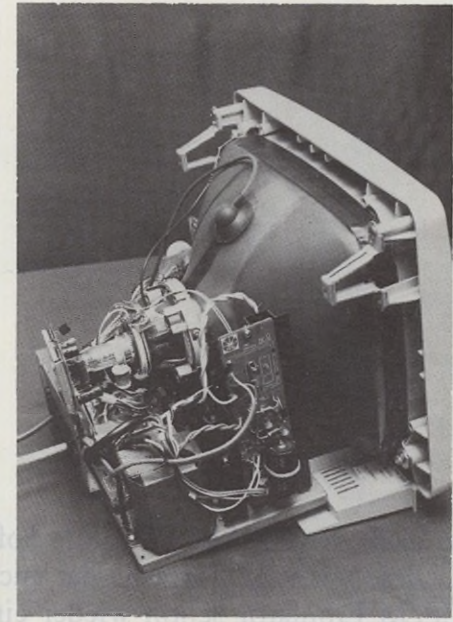
Software



Grafik

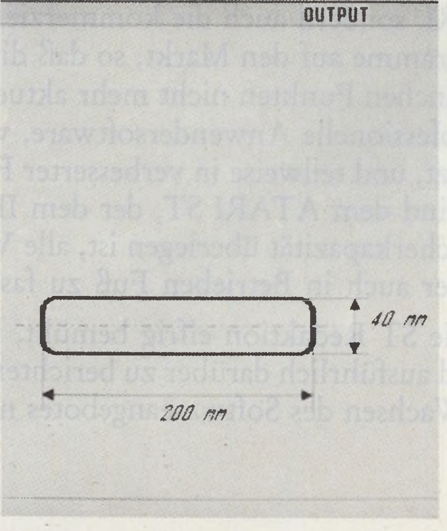
Bericht

Hardware



Listing

GEM



Kurse

CP/M

Aktuelles

Erstellung einer doppelseitigen Systemdiskette

Wer sich bis jetzt geärgert hat, daß er zwar ein doppelseitiges Laufwerk hat, aber das TOS sich nur von einer einseitig formatierten Systemdiskette laden ließ, dem ist nun geholfen. Die folgende Anleitung gibt allen Besitzern des ATARI Laufwerks SF 314 die Möglichkeit sich selbst eine doppelseitige Systemdiskette zu erstellen. Damit ist es möglich neben dem TOS auch noch jede Menge andere Programme oder auch Desk-Accessories unterzubringen. Für die Erstellung der neuen Systemdiskette wird allerdings ein Diskettenmonitor namens „Joshua" benötigt, den man sich aber bei jedem ATARI-Fachhändler kostenlos kopieren kann.

Wer schon weiß, was ein Diskettenmonitor ist, der kann jetzt den nächsten Absatz überspringen, da nun kurz dardauf eingegangen werden soll.

Ein **Diskettenmonitor** oder auch kurz Diskmonitor genannt ermöglicht den Inhalt einer Diskette in Form von hexadezimalen Zahlen bzw. ASCII-Zeichen anzusehen oder auch zu verändern. „Joshua" ist nun nicht nur ein solcher Diskmonitor, sondern man kann mit ihm noch einiges mehr, worauf wir hier aber nicht näher eingehen wollen. Er liest und schreibt immer ein **Langwort**, in dem vier hexadezimale Zahlen zusammengefaßt sind. Zum Beispiel „60384C6F", das erste Langwort aus unserem Bild 1.

Um nun einen Diskmonitor benutzen zu können, muß man noch wissen, daß jede Diskette beim Formatieren in sogenannte **Tracks** und **Sektoren** eingeteilt wird. Bei den ATARI-Laufwerken SF 354 und SFD 314 sind es 80 Tracks zu je 9 Sektoren. Ergänzenderweise muß man noch erwähnen, daß diese Laufwerke sogar 83 Tracks nutzen können, dies aber software-mäßig nicht vorgesehen ist.

Der Sektor, der uns nun interessiert, ist der Sektor 1 auf Track 0, der sogenannte **Bootsektor**. In ihm steht ein einleitender Ladesatz, der automatisch beim Formatieren generiert wird. Dieser Ladesatz dient dazu zu überprüfen, ob eine richtig formatierte oder intakte Diskette im Laufwerk ist. Außerdem wird der Ladesatz bei einem Reset oder Start des Systems in den Arbeitsspeicher geladen, um dort zu überprüfen, ob eine Systemdiskette mit dem TOS im Laufwerk ist. Das eigentliche Problem eine doppelseitige Systemdiskette zu erstellen, besteht nun darin, daß eben der richtige Bootsektor auf der neuen doppelseitig formatierten Diskette fehlt. Der Bootsektor wird nämlich nur dann mit-

```
Tr:$00 Se:$01 Again, Read, Write, Next, Previous, Boot, Modify, Copy, eXit
0000 60384C6F 61646572 A4833300 02020100 027000A0 05F80500 09000200 00000000
0020 00000000 00000004 00000000 8000544F 53202020 2020494D 470033FA FFE20000
0040 04823F39 00000446 3F3C0007 4E4D584F 4A806700 00F62A40 41FAFFD0 4A906606
0060 20890000 0432302D 0008E148 D0803840 D9FAFFB8 303AFFAA 67103C3A FFA6383A
0080 FFA4267A FFA26000 00B43C2D 000A382D 0008D86D 0006267A FF926100 00B26600
00A0 00AA204C 302D0006 E148E348 41F00000 43FAFF7C 90FC0020 B1CC6D00 008E700A
00C0 12300000 B2310000 66EA51C8 FFF47E00 1E28001B E14F1E28 001A2C7A FF4E267A
00E0 FF464284 BE7C0FF0 6C523607 5543C6ED 0002D66D 000CB87C 00406C08 4A44670E
0100 B6456710 61486642 E18CE38C D7C43C03 3A034284 D86D0002 DA6D0002 3407E24A
0120 D4471236 2001E149 12362000 08070000 6702E849 C27C0FFF 3E0160A8 4A446704
0140 610C6606 2F3AFEE0 4E754280 4E753F39 00000446 3F063F04 2F0B4267 3F3C0004
0160 4E4DDEFC 000E4A40 4E754E65 75746572 20426F6F 7465720D 0A284329 31393835
0180 20417461 72692043 6F72702E 000A0000 00000000 00000000 00000000 00000000
01A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000ED22
0000 `8Loader$.3.......p..x.........................TOS     IMG.3z^b..
0040 ..?9...F?<..NMXOJ.g..v*CAz^PJ.f..9...20-..aHP.8CYz^80:^*g.<:^&8:
0080 ^$&z^"`..4<-..8-..Xm..&z^.a..2f..* L0-..aHcHAp..Cz^|.|..1Ln...p.
00C0 .0..21..fjOH^t~..(..a0.(...z^N&z^FB.>|.pIR6.UCFm..Vm..8|.C1.JDg.
0100 6Eg.aHfBa.c.WD<...B.Xm..Zm..4.bJTG.6..aI.6.....g.hIB|..^>.`(JDg.
0140 a.f./:~`NuB.Nu?9...F?.?./.Bg?<..NM^|..JCNuNeuter Booter..(C)1985
0180  Atari Corp.....................................................
01C0 .............................................................n"
```

Bild 1: Geänderter Bootsektor

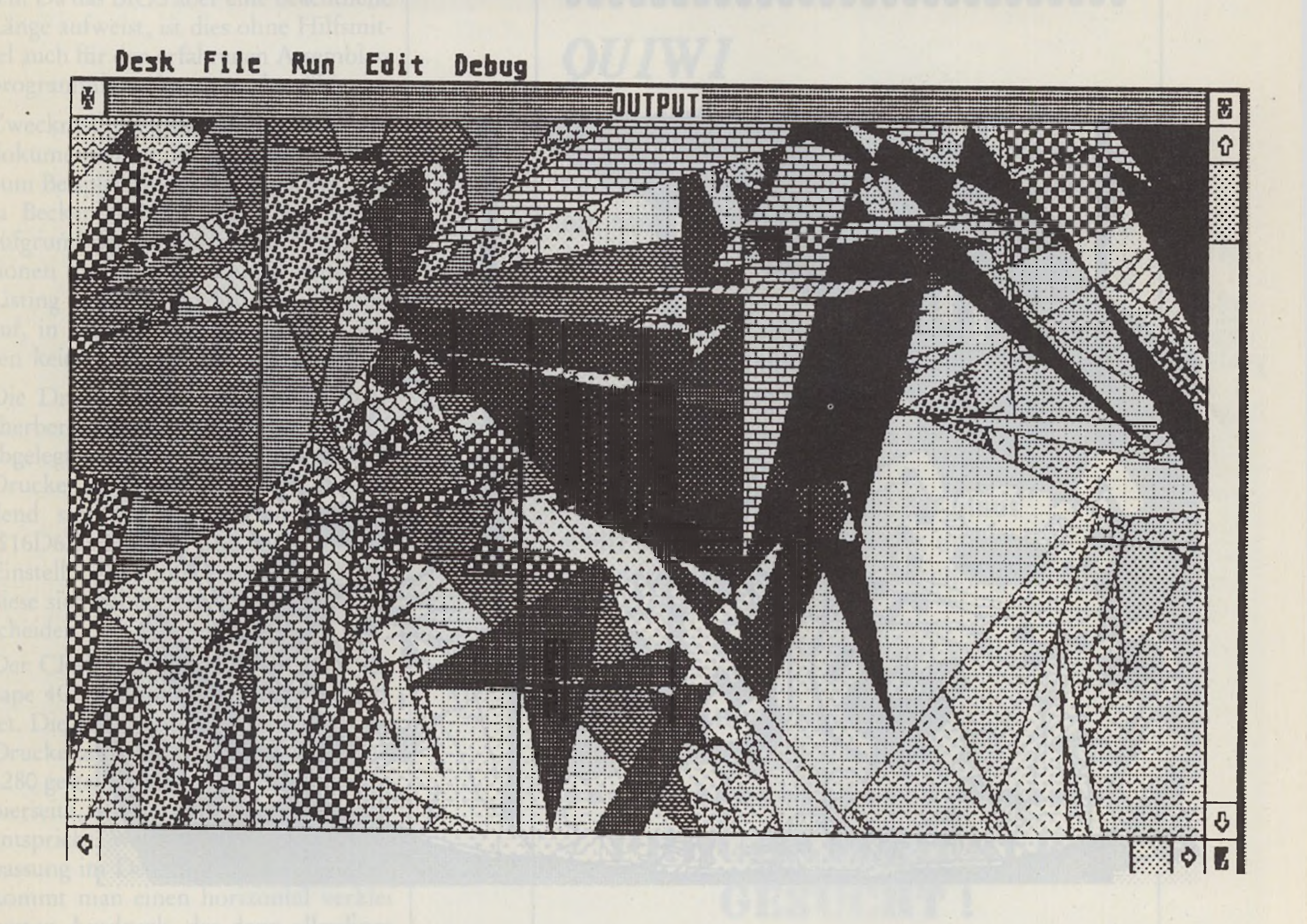
kopiert, wenn man eine ganze Diskettet auf einmal kopiert. Da aber von einer einseitigen (Original-Systemdiskette) auf eine doppelseitige neue Diskette kopiert werden muß, bekommt man eine Fehlermeldung, daß Original- und Zieldiskette im Format nicht übereinstimmen. Man muß also alle Dateien der Systemdiskette einzeln kopieren, was aber dann den Nachteil hat, daß der Bootsektor fehlt. Aus diesem Grund muß man jetzt „Joshua" benützen. Mit ihm wird der Bootsektor einfach von der einseitigen Systemdiskette eingelesen, auf die neue doppelseitige Systemdiskette kopiert, modifiziert und anschließend noch die Prüfsumme des Bootsektors berechnet und abgespeichert.

Die folgende Anleitung ist dazu genau zu befolgen.

1. Eine neue Diskette doppelseitig formatieren.
2. Alle Dateien der Original-Systemdiskette einzeln auf die doppelseitige Diskette kopieren.
3. „Joshua" laden und Taste drücken. Danach erscheint eine Kurzübersicht der „Joshua"-Befehle.
4. Taste „E" für die Benutzung des Diskmonitors drücken.
5. Originalsystemdiskette in das Laufwerk einlegen.
6. Taste „R" für READ (Lesen) drücken.
7. Track: 0 Sektor: 1 eingeben.
8. Doppelseitige Diskette mit TOS-Dateien in Laufwerk einlegen.
9. Taste „W" für WRITE (Schreiben) drücken.
10. Track: 0 Sektor: 1 eingeben.
11. Mit Taste „Y" für YES bestätigen, daß der Bootsektor auf die neue Systemdiskette geschrieben werden soll.
12. Taste „M" für MODIFY (Modifizieren) drücken.
13. Mit Taste „D" den Cursor nach rechts auf das fünfte Langwort setzen und „Return" drücken. (Die übrigen Cursorbewegungen liegen auf den Tasten „S" für links, „E" für oben und „X" für unten.)
14. Jetzt erscheint am oberen Bildschirmrand das alte Langwort (OLD VALUE) „027000D0". Hinter NEW VALUE jetzt „027000A0" eingeben und mit „Return" bestätigen.
15. Das 6. Langwort von „02F80500" auf „05F80500" und das 7. Langwort von „09000100" auf „09000200" wie oben beschrieben umändern.
16. Ende der Änderung mit Taste „Esc".
17. Mit Taste „Y" für YES das Schreiben der Änderung auf Diskette bestätigen.
18. Taste „B" für „Boot erstellen" drücken.
19. Die Frage „Boot COMMAND. PRG first" mit „N" für NO verneinen.
20. Die Frage „Calculate Boot check sum" mit „Y" für YES bejahen.
21. Neue Prüfsumme mit „Y" für YES auf die Diskette schreiben.
22. Mit Taste „X" für EXIT (Ausgang) ins Ausgangsmenü.
23. Durch gleichzeitiges Drücken von „Control" und „C" wieder in den Desktop zurückkehren.

Wenn diese Anleitung genau befolgt wurde, ist man jetzt im Besitz einer doppelseitigen Systemdiskette. Ob alles geklappt hat, kann man leicht durch einen Neustart des Systems überprüfen.

Kontrolliertes Chaos

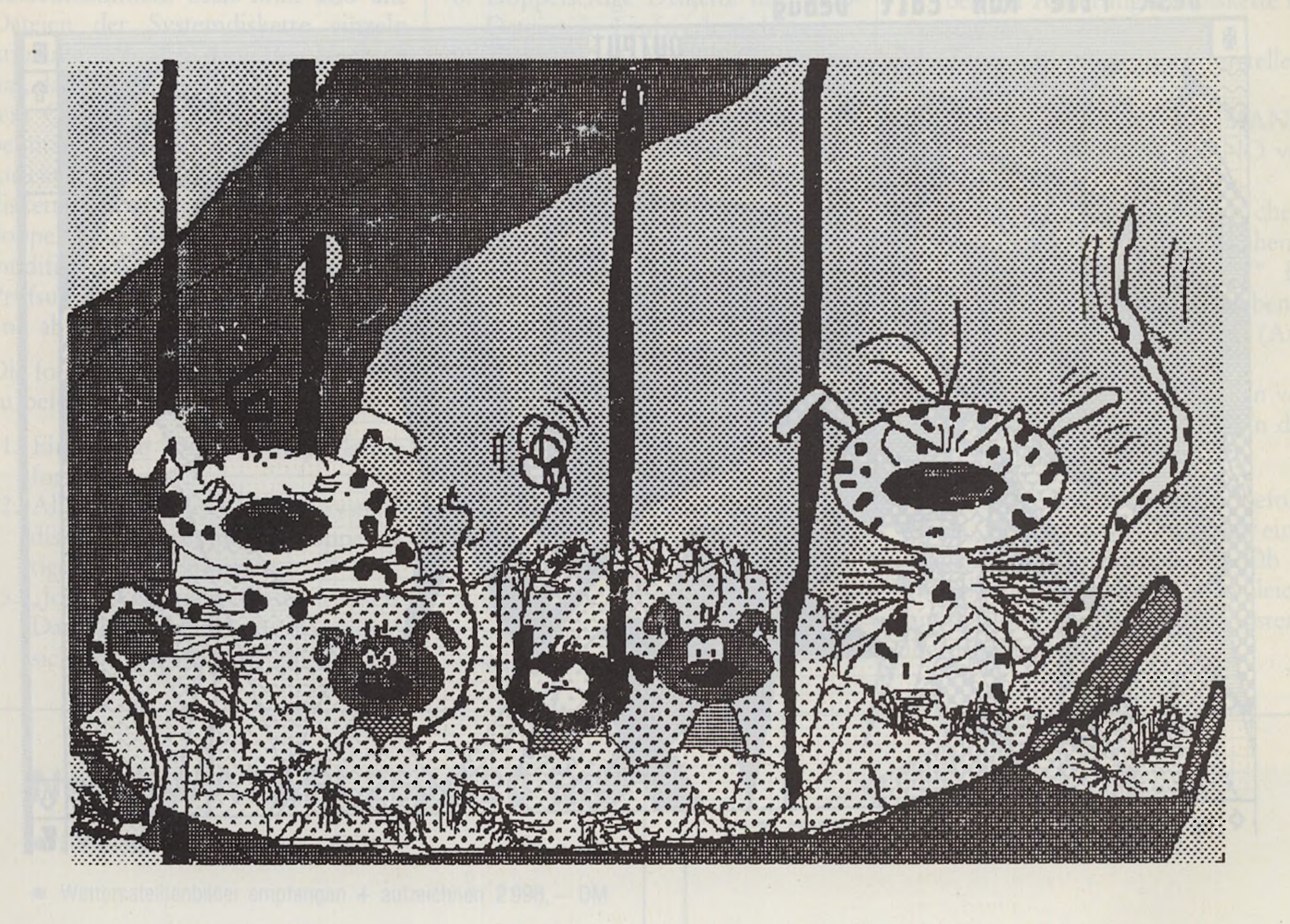


Mit diesem Programm läßt sich ein abstraktes Bild erzeugen. Obwohl es eher nach einem mißglückten Programmierversuch aussieht, ist das Ganze doch zu steuern. Es wird hierbei eine Besonderheit des CIRCLE bzw. des PCIRCLE-Befehls ausgenutzt, der normalerweise für Anfangs- und Endwinkel Werte zwischen 0 und 3 600 (also Grad ★ 10) erwartet. Bei Werten über 3 600 fängt er wieder am Nullpunkt an. Erreicht der Winkel nun aber Werte über 32 768, so fühlt sich der interne Kreis-Algorithmus dazu veranlaßt, wunderschöne bizarre Muster auf dem Bildschirm zu entwerfen. Verwendet man den PCIRCLE-Befehl, so wird der entstehende Körper auch ausgemalt, wobei man einen Eindruck von der hohen Geschwindigkeit bekommt, mit der das Basic Flächen ausmalt. Das Programm legt den Anfangswinkel mit 32 768 fest und erhöht ihn bei jedem Durchgang. Erreicht der Winkel Werte die größer als 65 536 sind, so verhält sich der Befehl wieder eine Weile normal. Da wir das hier natürlich vermeiden wollen, setzen wir ihn wieder auf den Anfangswert. Die Koordinaten des PCIRCLE-Befehls werden mit Hilfe der Zufalls-Funktion bestimmt und plazieren den Ursprung dieses doch sehr künstlerisch angehauchten Grafikbefehls über den gesamten Bildschirm. Durch Änderung der Befehlsparameter können viele andere interessante Bilder erzeugt werden. Hier sei zum Beispiel empfohlen die x/y-Koordinaten festzulegen oder den Radius zu verkleindern.

List of CHAOS.BAS

```
100 '          Kontrolliertes C H A O S
110 '                            by Harald
120 '
130 FULLW 2: CLEARW 2: CLOSEW 3
140 wa=32768
150 start:
160 COLOR 1,1,1,B,2
170 x=rnd(1)★500
180 y=rnd(1)★400
190 r=rnd(1)★300
210 PCIRCLE x,y,r,wa,wa+200
220 wa=wa+rnd(1)★500
230 b=b+1 : if b>24 then b=1
240 if wa>65536 then wa=32768
250 goto start
```

Hardcopy mit CP80 oder ähnlichen Druckern



Ein Manko einiger Drucker war bisher, daß sie bei der eingebauten Hardcopy-Funktion des ATARI ST nicht funktionierten. Bei Anwählen der 'Alternate Help'-Funktion sah sich der Drucker dazu veranlaßt entweder nur Zeichensalat, wildes Gepiepse und Formfeeds auf das Papier zu bringen oder eine Hardcopy auszugeben, die horizontal gestaucht war und größere Zwischenräume aufwies.

Die Ursache war klar. Diese Drucker verstehen andere Bitmap-codes als IBM oder Epson-kompatible Drucker.

Zur Anpassung gibt es verschiedene Möglichkeiten. Man kann eine neue Druckerroutine schreiben die speziell auf den verwendeten Drucker angepaßt ist. Dies ist allerdings ein komplizierter Weg, der zwar mit jedem Drucker funktioniert, sei er auch noch

Die Anpassung kann mit folgendem Basic Programm erfolgen:

```
10 'Druckeranpassung CP 80 oder ähnliche Drucker
20 poke &h16D76,&h3315        ' Linefeed auf 15/216"
30 poke &h16D72,&h4CFF        ' Bitmap normale Größe
40 poke &h16D6E,&h4CFF          Bitmap verkleinert
50 ' poke &h16D6E, &h4BFF Bitmap vergrößert (Hälfte des Bildschirms)
60 ' poke &h16D72, 8h4BFF  Bitmap vergrößert (2/3 des Bildschirms)
70 poke &h88D4,0                Linefeed ausschalten
80 poke &h8924,0              ' Linefeed ausschalten
90 print "Drucker angepasst"
```

so ungewöhnlich. Bei den meisten Druckern ist aber ein wesentlich einfacherer Weg praktizierbar, nämlich eine Änderung im BIOS. Hierfür muß man den Teil des BIOS, der die Druckersteuerung beinhaltet, anpassen. Da das BIOS aber eine beachtliche Länge aufweist, ist dies ohne Hilfsmittel auch für den erfahrenen Assemblerprogrammierer kaum zu bewältigen.

Zweckmäßigerweise benötigt man ein dokumentiertes BIO-Listing, wie es zum Beispiel im 'Intern' Buch von Data Becker enthalten ist. Zwar treten aufgrund der verschiedenen TOS-Versionen einige Unterschiede zwischen Listing und der eigentlichen Version auf, in unserer Anwendung aber waren keine Unterschiede vorhanden.

Die Druckersequenzen sind im Speicherbereich von $16D5E bis $16D87 abgelegt und können auf den eigenen Drucker angepaßt werden. Entscheidend sind hierbei die Bitmapcodes ($16D6D; $16D71) und der Code zur Einstellung des Linefeeds ($16D75), da diese sich bei einigen Druckern unterscheiden.

Der CP80 zum Beispiel wird mit 'Escape 4C' auf hohe Auflösung geschaltet. Die Hardcopy geht damit, falls die Druckeranpassung im Desktop auf 1280 gestellt wurde, über die ganze Papierseite, was der normalen Hardcopy entspricht. Wenn man die Druckeranpassung im Desktop auf 960 stellt, bekommt man einen horizontal verklei nerten Ausdruck, der dann allerdings nicht mehr maßstabsgerecht ist.

Falls man nun in die beiden Speicherzellen, die den Bitmapmodus einschalten (siehe Programm) eine 'Escape 4B' schreibt, entstehen horizontal gedehnte Hardcopies, die sich je nach eingestellter Druckerauflösung im Desktop, unterscheiden. Eine solche Ausschnittsvergrößerung bildet dann natürlich nur noch einen Teil des linken Bildschirmes ab, ist aber für den Einzelfall recht interessant.

Auf diese Art und Weise kann man auch auf Druckern, die ohne diese Anpassung eine korrekte Hardcopy zu Papier bringen, eine Veränderung des Hardcopyformates erreichen.

Viel Erfolg beim Experimentieren.



Alles in Einem!

Der Makro-Assembler von KUMA

Assembler ist ein Begriff, der leicht Schrecken erweckt und das nicht ohne Grund, denn die Beherrschung von Assembler gehört zu der höheren Schule der Programmierung.

Warum in Assembler programmieren? Die Frage ist insofern berechtigt, daß es auch andere Sprachen gibt, wie z. B. C oder FORTH, die „fast" so schnell wie Assembler Programme sind. Es existieren aber Bereiche, wie bei der Graphikherstellung, gewählten Problemen bei der Verarbeitung von großen Mengen an Daten, in der Messungs- und Steuerungstechnik usw., in denen der Schnelligkeitsfaktor eine große Rolle spielt. Eingriffe, Änderungen und Erweiterungen im Betriebssystem sind in der Regel nur in Assembler durchführbar.

Erfahrungsgemäß wird jeder intensive Programmierer sich irgendwann mit dem Maschinen-Code seines Rechners beschäftigen. Für jene ist ein Assembler ein unentbehrliches Werkzeug.

KUMA bietet für alle ST-Besitzer ein Assembler-Paket, welches sehr kompakt aufgebaut ist.

Der Editor und das Editieren

Um ein Programm herzustellen, braucht man einen Text-Editor, mit dem das Programm wie ein normaler Text geschrieben wird. In der Regel ist ein Editor einem Textverarbeitungsprogramm sehr ähnlich, wobei die Unterschiede nur in dem Komfort des letzteren liegen.

Normalerweise gehört zu einem Assembler-Paket ein Editor, sowie andere Hilfsprogramme, die nach Bedarf einzeln nachgeladen und ausgeführt werden. Bei KUMA ist das anders, Editor, Assembler, sowie Debugger werden im Rechner einmal geladen und bleiben erhalten.

Der Editor des KUMA Assembler-Pakets ist ein sogenannter Zeilen-Editor, und damit gehört er zur Prähistorie der Editierung. Nach Eingabe von „I" wird automatisch eine Zeilen-Nummer generiert, die nicht beliebig zu ändern ist. Dann stehen auf dieser Zeile beide Cursorbewegungen, nach links und rechts, zur Verfügung. Die Delete-Taste, sowie Backspace, dienen zum Löschen eines Zeichens in Bezug auf die Cursor-Position. Jede Zeile wird durch RETURN abgeschlossen und übernommen. Gleichzeitig besteht aber auch noch die Möglichkeit mit „Control P" anstatt des RETURN alle Ausgaben des Assemblers nicht auf den Bildschirm, sondern auf einen Drucker auszugeben.

Möglichkeiten den Cursor nach oben oder unten frei zu bewegen, existieren nicht, genausowenig wie das Einrücken von Textblöcken. Das Reeditieren läuft unter großem Aufwand ab: Durch den Befehl TARGET-Zeilennummer wird der Cursor auf die gewünschte Zeile verlagert; danach kann man mit EDIT-Zeilennummer die Zeilen verbessern oder verändern.

Man muß zugeben, daß ein solcher Editor für längere Programme nicht geeignet ist, bei kürzeren Programmen ist man allerdings dankbar, daß das Nachladen von Editor-Assembler, Linker usw. entfällt. Wenn man sich bessere Editierungsmöglichkeiten wünscht, kann man sich durch das Anwenden eines besseren Editors (z. B. der von METACOMCO) oder durch ein normales Textverarbeitungssystem (z. B. SM-Text) helfen.

Der Assembler

Der Assembler von KUMA läßt kaum Wünsche offen. Er erkennt alle Stan-

```
Andelos 68000 Assembler SEKA v1.1 - Copyright (C) Andelos Systems 1984-1985
Atari ST version - Sole distributors: Kuma Computers, Pangbourne, Berks, UK

SEKA>n $fc0008
FC0008 MOVE.W  #$2700,SR
FC000C RESET
FC000E CMP.L   #$FA52235F,$FA0000.L
FC0018 BNE     $FC0024
FC001A LEA     $FC0024(PC),A6
FC001E JMP     $FA0004.L
FC0024 CMP.L   #$31415926,$000426.L
FC002E BNE     $FC004C
FC0030 MOVE.L  $00042A.L,D0
FC0036 TST.B   $00042A.L
FC003C BNE     $FC004C
FC003E BTST    #$0000,D0
SEKA>x
D0=00000000 00000000 00000000 00000000  D4=00000000 00000000 00000000 00000000
A0=00000000 00000000 00000000 00000000  A4=00000000 00000000 00000000 00047ED6
SSP=00047EEA USP=00047ED6 SR=0000  PC=045B60 ILLEGAL
SEKA>
```

Der disassemblierte Anfang des Betriebssystems

dard-Motorola-Mnemoniks des 68000er Prozessors. Damit können Programme, die für den Prozessor mit einem anderen Assembler geschrieben worden sind, und soweit sie nicht systemabhängig sind, ohne weiteres übernommen werden. Das Eingabeformat ist nicht festgelegt und dem Programmierer freigestellt. Das bedeutet, daß nicht überall mit Blanks angeordnet werden soll, das verrichtet der Assembler selbst.

Eine Eingabe kann so aussehen:

MOVE. B #147, D2
oder
EIN: MOVE. L #$OA, D0

Die einzige Einschränkung besteht nur darin, daß eine Eingabe einen Assembler-Sinn haben muß, so, wie die obigen Beispiele. Die nächste Notation wäre ein unsinniger Text:

MOVE. W (A1)

Diese Eingabe wird von dem Editor übernommen, aber für den Assembler fehlt ein Operand und er weiß nicht, was er mit ihm anfangen kann, so daß eine Fehlermeldung resultiert.

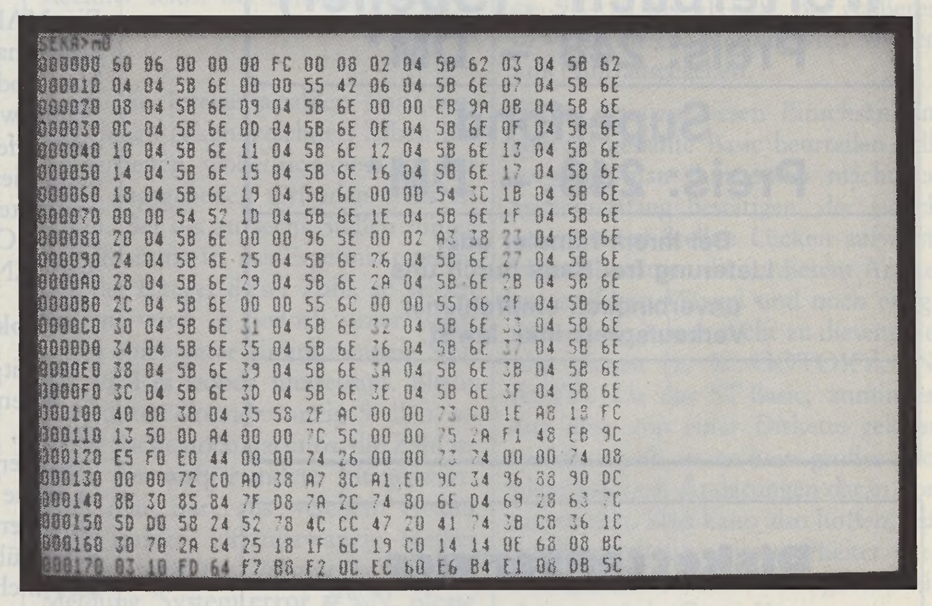
Der Assembler akzeptiert und verarbeitet Labels beliebiger Länge, die eine Mischung von Buchstaben und Zahlen sein können, wobei die einzige Einschränkung darin besteht, daß das erste Zeichen immer ein Buchstabe sein muß. Die Labels werden von einem Assembler-Befehl durch Doppelpunkt (:) getrennt.

Nach einem Operanden-Feld kann man beliebige Kommentare hinzufügen. Diese werden von dem Operanden-Feld durch ein Semikolon (;) getrennt. Numerische Ausdrücke sowie mathematische und logische Operationen stehen zur Verfügung.

Eine Reihe von Pseudo-Operatoren kommen dem Programmierer zu Hilfe. Sie dienen in den meisten Fällen zur Steuerung des Assembler-Listings, sowie zur einfachen Generierung von Tabellen. z. B. folgende Pseudo-Operatoren:

Tabelle = $ 7FD00
oder
DC.L $ 70000,....

Außer der bedingten Assemblierung verfügt der KUMA Assembler über eine sogenannte direkte Assemblierung. Nach Eingabe von „A" (für Assembler), gefolgt von einer Adresse beginnt der Assembler direkt zu interpretieren sowie auszuführen. Dabei sind Kommentare wie Labels nicht mehr möglich. Bei der bedingten Assemblierung stehen verschiedene Optionen zur Wahl: Durch Option „O" werden Verzweigungen optimiert. Das Auswählen von „L" veranlaßt den Linker einen verknüpfbaren Code zu erzeugen. Bei „E" oder „P" wird ein formatiertes Listing aus dem Drucker geschickt. Eine Auflistung auf dem Bildschirm findet durch die Option „V" statt. Dabei können auch mehrere Optionen miteinander vermischt werden.



Hexadezimales Auflisten

Der Linker

Normalerweise generiert ein Assembler einen verknüpfbaren Code, der später durch ein Hilfs-Programm, dem sogenannten Linker, in einen ausführbaren Code umgewandelt wird. Bei dem KUMA Assembler ist dies etwas anders. Der Assembler erzeugt ein Programm, das sofort ausführbar ist. Der Linker seinerseits, erzeugt ein Programm, das zum Teil optimiert worden und deswegen schnell ist. Bei kurzen Programmen kann man auf den Linker ganz verzichten.

Die „L" Option in Assembler-Modus erzeugt ein „gelinktes„ Programm. Dabei ist zu beachten, daß der Linker nur mit absoluten, nicht verschiebbaren Werten arbeitet.

Der Debugger

Bei der Programmierung von höheren Sprachen unterlaufen häufig Fehler, die nur nach einer langen Suchaktion zu finden sind. Bei der Assembler-Programmierung ist das auch nicht anders, nur daß hier das Auffinden eines Fehlers um einiges problematischer ist als in BASIC. Hier schafft nur ein Debugger Abhilfe. Das Assembler-Paket, das uns heute beschäftigt, besitzt ein durchaus komfortables „Fehlersuchprogramm".

Alle existierenden Register des 68000er Prozessors werden durch die Option „X" angezeigt. Mit „X", plus der Angabe eines bestimmten Registers wird dessen Inhalt angezeigt und dessen Veränderung ermöglicht. Suchfunktionen, sowie Einzelschrittabläufe eines Programms sind auch möglich. Ganze Speicherbereiche können entweder gefüllt oder kopiert werden. Multiabbruchpunkte kann man beliebig setzen um den Ablauf eines Programms zu beobachten.

Assembler mit MAKRO-Verarbeitung

Der KUMA-Assembler bietet die Möglichkeit der Erstellung von MAKROS, die ohne Zweifel dem Programmierer, sowie der Lösbarkeit eines Programms, zu Hilfe kommen. Ein MAKRO bildet eine bestimmte Befehlssequenz, die fortwährend wiederholt wird, und nicht eine Verzweigung im Programm, wie es der Fall bei einem Unterprogramm wäre. Deswegen wird ein Programm, das MAKROS verwendet, immer länger, aber in der Regel schneller.

Ein MAKRO wird durch eine Kopfzeile charakterisiert, wo immer ein Name oder ein Symbol erscheint. Der Name wird durch Doppelpunkt von dem Befehl MAKRO getrennt. Später kann dieser mit verschiedenen Lokalparametern versehen werden. Eine MAKRO-Definition wird durch den Befhl ENDM beendet.

Die Dokumentation

Das mitgelieferte Handbuch umfaßt 30 Seiten, in denen die verschiedenen Befehle, sowie Fehlermeldungen sehr mager erklärt sind. Dieses Handbuch ist keine Hilfe (muß auch nicht sein) für einen Anfänger, der sich mit der Befehlsübersicht des 68000er Prozessors nicht auskennt. Hier muß man sich ein spezielles Buch über die Technik der Programmierung dieses Prozessors besorgen, wie es auch von KUMA empfohlen wird. Zwei Beispiele sollen den Umgang mit dem Assembler verdeutlichen.

Das ist für den erfahrenen Programmierer ausreichen. Für diejenigen, die noch nicht erfahren genug sind, wären ein paar Beispiele mehr angebrachter, aber das würde den Rahmen dieses Tests sprengen.

Was uns an dem KUMA Assembler gefällt, ist, daß es sich um einen kompakten Einpaß-Assembler handelt, mit angenehmen Möglichkeiten zur Fehlersuche, sowie der Herstellung von MAKROS. Der schwache Punkt wäre der einfache Zeilen-Editor. Aber wo bekommt man alles auf einmal und dazu noch perfekt.

ST-BASIC

Vergleich von Genauigkeit und Geschwindigkeit

Genauigkeit bei der Zahlendarstellung

Wenn man mit einem Computer arbeitet geht man meist davon aus, daß die Ergebnisse, die man bei Rechenoperationen erhält stimmen. Wenn man sich näher mit dem Computer beschäftigt und wie er Rechenoperationen durchführt, dann versteht man, daß dies nicht immer der Fall sein kann. Während Taschenrechner meist mit BCD-Zahlen, also dezimal, rechnen wandeln Computer im allgemeinen eingegebene Dezimalzahlen in Dualzahlen um. Bei dieser Umwandlung entstehen Rundungsfehler, weil der Rechner nur eine begrenzte Anzahl von Stellen für die Mantisse zu Verfügung stellt. Das ATARI ST BASIC stellt bei einfacher (normaler) Genauigkeit nur **sechs** Stellen zur Verfügung. Wenn man Variablen als 'Doppelgenaue' definiert, erhält man **neunstellige** Ergebnisse. Doch ist dabei nur die siebte Ziffer noch eine 'Echte'. Die beiden anderen scheinen durch Zufall zu entstehen. Hier ein paar Beispiele:

Eingabe:	Ausgabe: einfach-	doppeltgenau
123456	123456	123456
1234567	1.2345E+06	1234567.04
100000.9	100000	100000.9
999999★10	9.9998E+06	9999989.76
999.999	999.998	999.998976
9.9	9.89999	9.89999936
3.1415	3.14149	3.14149984

Tabelle 1

Man muß diesem Basic deshalb eine schlechte Genauigkeit anlasten, zumal selbst doppeltgenaue Variablen nur magere sieben relevante Stellen liefern. Der IBM AT kann hier zum Beispiel 7 bzw. 17 Stellen aufweisen und selbst der Commodore 64 bringt 8 Stellen, ohne die Möglichkeit der doppelten Genauigkeit zu besitzen. Die Ungenauigkeiten, die sich daraus ergeben, sollten jedem Programmierer bzw. Anwender klar sein, weil er sonst Ergebnisse oder Fehler nicht erkennen bzw. richtig beurteilen kann. Die einfachen Beispiele aus Tabelle 1 verdeutlichen in ausreichender Weise, daß der Rechner selbst bei einfachsten Zahlenausgaben keine exakte Darstellung liefert.

Auch die Integervariablen (mit dem %-Zeichen gekennzeichnet) haben Besonderheiten, wobei sich diese aus den oben angegebenen Gründen erklären lassen. Bei bis zu sechs Stellen hinter dem Komma ist das Ergebnis tatsächlich der Integer dieser Zahl, ab sieben Stellen wird es jedoch interessant. Bleibt man unter einundzwanzig Stellen wird es jedoch interessant. Bleibt man unter einundzwanzig Stellen, so ist die Sache noch harmlos, die Zahl ist dann der **aufgerundete** Integerwert. Gibt man nun, aus welchem Grund auch immer, einundzwanzig Stellen ein, so wird das Programm mit der Meldung 'System **error #%N**, please restart at line...' abgebrochen. Hier ist also Vorsicht geboten und auch bei zweiundzwanzig Stellen und mehr, denn dann lautet das Ergebnis **Null**.

Die angesprochenen Fehler werden sicherlich nur selten auftreten, denn wer gibt schon so viele Stellen ein wenn der Rechner nur sechs oder sieben benutzt. Aber etwas ungewöhnlich ist es schon, und andere Basic-Versionen schaffen es doch auch annehmbare Ergebnisse zu liefern.

Zu guter letzt noch ein Beispiel zu einer einfachen Subtraktion.

1.01−0.01−1 = 0
1.01−1−0.01 = −9.31322E−09

Auch hier ist das Ergebnis bemerkenswert, wenn auch der Fehler mit rund 1E−10 nur sehr gering ist, so zeigt es doch, daß man auch damit vorsichtig umgehen sollte und bei dieser Rechnung z. B. keine Abfrage nach Null machen sollte.

Eine weitere Besonderheit ist der Wertebereich der darstellbaren Zahlen. Er reicht nicht, wie allgemein üblich, von 1E−38 bis 1E+38, sondern nur von ca. 9E−20 bis 5E+18. Bei größeren bzw. kleineren Werten wird wiederum Null ausgegeben.

Wenn man zu diesen Einschränkungen das gesamte Basic beurteilen soll, so muß man ihm einen mächtigen Sprachumfang bestätigen, der jedoch an manchen Stellen Lücken aufweist. Hinzu kommen die in diesem Artikel aufgezählten Mängel und noch einige andere, die jedoch nicht zu diesem Gebiet passen (z. B. GOTOXY, INKEY$). Da das ST-Basic, zumindest zur Zeit, von einer Diskette geladen werden muß, ist es kein großes Problem jederzeit Änderungen daran vorzunehmen. Man kann also hoffen, daß an diesem Basic weitergearbeitet wird und daß dann ein wirklich gutes Basic daraus wird. Gute Voraussetzungen sind dafür zur Genüge gegeben.

Geschwindigkeit des ST-BASIC

Um die Geschwindigkeit des Basic's bewerten zu können muß man die Ausführungszeit einzelner Befehle feststellen. Diese kann man dann mit den Zeiten anderer Rechner vergleichen und bewerten. Da die Ausführungszeit im Bereich von einigen Millisekunden liegt, muß man den einzelnen Befehl mehrmals ausführen um die Zeit bestimmen zu können. Dazu wird eine einfache Schleife (siehe Listing) be-

nutzt. Für die Ausführung dieser (Leer-) Schleife benötigt das Programm 8,7 Sekunden. Diese Zeit muß nun bei jeder weiteren Zeitmessung abgezogen werden um die Zeit für das 10 000malige Ausführen des Befehls zu erhalten. Teilt man dieses Ergebnis durch 10, so erhält man die Ausführungszeit für einen Befehl in Millisekunden. Die Ausführungszeiten werden denen des GW-Basic auf dem IBM-kompatiblen Commodore PC 10 gegenübergestellt (siehe Tabelle 2). Wie man sieht ist der Interpreter des ST-Basic sehr schnell. Es gibt fast keinen Unterschied zwischen den Grundrechenarten und den Funktionen SIN, SQR und LOG. Sie benötigen alle nur rund 1,5 Millisekunden für ihre Ausführung. Das GW-Basic ist zum Teil deutlich langsamer. Es benötigt schon für die Grundrechenarten 2 ms. Bei den Funktionen Sinus und Cosinus wird es dann besonders deutlich. Das ST-Basic ist hier mehr als 10mal so schnell. Bei den Funktionen LOG, ATN und SQR holt das GW-Basic zwar wieder auf, doch es erreicht auch hier nicht die Geschwindigkeit des ATARI-Basics. Erst wenn es um die Bildschirmausgabe geht zeigt das ST-Basic Schwächen. Zum Testen sollen die Zahlen 1 bis 1 000 mit einer einfachen Schleife auf dem ganzen Bildschirm ausgegeben werden. Geschieht die Ausgabe mit 'PRINT A', so benötigt das Programm dafür 3 Minuten und 35 Sekunden. Dies ist eine beachtlich lange Zeit. Augenscheinlich hat das ST-Basic Schwierigkeiten mit dem 'Scrollen' des Bildschirms. Benutzt man nämlich statt des angesprochenen Befehls die Anweisung 'GOTOXY 1,1 : PRINT A', so werden dafür nur 24 Sekunden benötigt. Das GW-Basic ist hier eindeutig im Vorteil (siehe Tabelle 3).

```
1 FULLW 2 : CLEARW 2
2 X=0 : Y=9
3 PRINT "START !"
4 FOR A=1 TO 10000
5 ...
6 NEXT A
7 PRINT "STOP !"
8 END
9 RETURN
```

Listing 1 zu Bestimmung der Ausführungszeiten

Befehl	Zeit in ms ST-BASIC	GW-BASIC
(Leerschleife	0,87	1.14)
B=0	0.7	1.2
B=9	0.9	1.1
B=9.9E−20	0.7	1.0
B=X	0.7	1.2
B=X+9	1.4	2.0
B=X−9	1.4	2.0
B=X★9	1.4	2.0
B=X/9	1.4	2.0
B=X/Y	1.2	2.0
B=SIN(Y)	1.5	19.4
B=COS(Y)	1.5	24.3
B=TAN(Y)	2.2	41.0
B=ATN(Y)	5.9	5.4
B=LOG(Y)	1.6	7.0
B=LOG10(Y)	2.4	2.6
B=SQR(Y)	1.2	2.6
B=Y 2	3.0	4.1
B=EXP(Y)	1.7	5.0
B=SGN(Y)	1.3	1.7
B=PEEK(Y)	1.5	2.3
SWAP X,Y	0.9	0.7
IF X>Y THEN 6	1.0	2.1
IF X<Y THEN 6	1.3	2.1
REM ABCDEFGHIJ	0.2	0.6
GOTO 6	0.4	0.4
GOSUB 9	0.8	0.9
GOSUB programm	0.7	
CLEARW/CLS	600.0	270.0
o. Fußzeile		120.0
E$=A$	1.4	0.7
E$=B$	0.8	0.6
E$=A+B	4.6	1.8
E=RIGHT$(A$,9)	2.1	2.1
E$=MID$(A$,5,3)	2.6	2.5
X=INSTR(1,A$,"9"	2.4	3.1

(Anm.: a$="1234567890",b$="1")

Tabelle 2

	Zeit in min.	
PRINT A	3:35	0:42
PRINT A,	1:20	0:37
PRINT A;	0:35	0:19
GOTOXY 1,1:? A	0:24	
LOCATE 1,1:? A		0:22

Tabelle 3

	ATARI ST	PC 10
Prozessor	68 000	8 086
Arbeitsweise:		
extern	16 Bit	8 Bit
intern	32 Bit	16 Bit
Taktfrequenz	8 MHz	4,77 MHz

Tabelle 4 Daten der CPUs

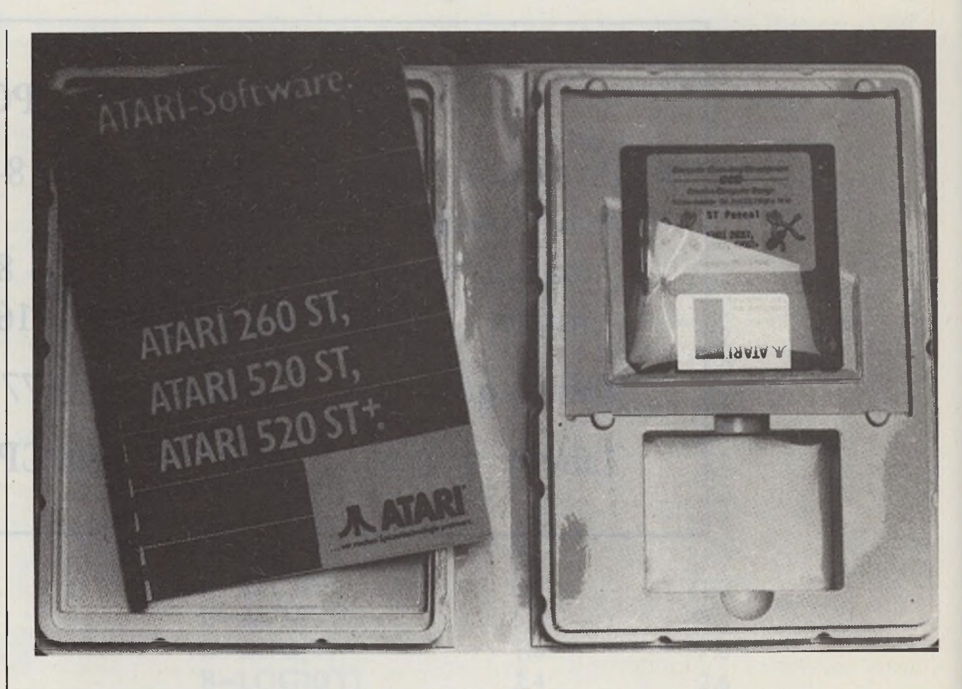
Bei der Stringverarbeitung liegen die Zeiten beider Rechner im gleichen Berich. Allerdings hat das GW-Basic auch hier einen Vorsprung.

Anhand eines kleinen Programms läßt sich nun zeigen, daß die so ermittelten Werte tatsächlich eine Auswirkung auf die Programmablaufgeschwindigkeit haben. Dazu wird ein Primzahlenprogramm verwendet, mit dem die Primzahlen bis zum Wert 1 000 berechnet werden. Das ST-Basic benötigte dafür 18 Sekunden, das GW-Basic immerhin 30 Sekunden. Würde man allerdings die Werte auch noch auf dem Bildschirm ausgeben wollen, dann würde sich das Verhältnis zugunsten des GW-Basic verschieben.

Zusammenfassend läßt sich sagen, daß das ST-Basic einen ausgewogenen Eindruck macht. Die Ausführungszeit liegt bei fast allen Befehlen in der gleichen Größenordnung. Es übertrifft damit das GW-Basic in fast allen Bereichen. Damit könnte man nun zufrieden sein, doch wenn man sich die Daten der verwendeten CPUs (Tabelle 4) ansieht ist man doch ein wenig enttäuscht. Der ATARI ST ist mit dem 68 000-Prozessor ausgerüstet. Dieser Prozessor kann extern 16 Bit und intern 32 Bit verarbeiten und ist zudem noch mit 8 MHz getaktet. Der PC 10 hat dagegen mit dem 8086-Prozessor nur 8 bzw. 16 Bit zur Verfügung und ist auch nur mit 4,77 MHz getaktet. Nimmt man diese Daten zur Bewertung hinzu, so muß man feststellen, daß das ST-Basic die Fähigkeiten der schnellen CPU nicht ausreichend nutzt. Besonders bei der Stringbehandlung und dem 'Bildschirmrollen' ist der Basic-Interpreter einfach zu langsam. Wie auch bei der Genauigkeit wäre deshalb ein überarbeitetes Basic wünschenswert.

(MN)

ST Pascal Compiler von CCD



ST Pascal wird komplett auf einer Diskette und mit einem 45 Seiten 'dicken' Handbuch geliefert, das dem Kunden auch gleich auf den ersten Seiten klarmacht, für welchen Kundenkreis es konzipiert ist. Der Pascal-Neuling sollte gar nicht erst versuchen, sich da hindurch zu kämpfen. Der Kauf eines ordentlichen Pascal-Lehrbuchs ist hier nicht zu umgehen.

Der Pascal Compiler ist mit einem Kopierschutz versehen. Es läßt sich zwar ein Backup erstellen welches aber nur in Verbindung mit der Originaldiskette lauffähig ist. Ein solches Backup stellt eine ausreichende Sicherheit dar und ist als eine akzeptable Lösung eines Kopierschutzes anzusehen. Für den durchschnittlichen Benutzer des Atari ST, der nur über ein Diskettenlaufwerk verfügt, kommt nun der nächste Arbeitsschritt vor der eigentlichen Anwendung des Pascal-Systems: Man mache aus der Originaldiskette eine Compiler- und eine Linkerdiskette. Dies ist zum Glück nur einmal vonnöten und bleibt dem Besitzer eines zweiten Laufwerkes erspart.

Der Editor

Wenn dies alles vollbracht ist, kann es wirklich losgehen: Man legt die Compiler-Disk ein und lädt den vorhandenen Editor. Dessen Funktionen sind für die ordentliche Erstellung eines Pascal-Source Textes vollkommen ausreichend, wobei der automatische Tabulator zur übersichtlichen Gestaltung des Quelltextes sehr hilfreich ist. Nach der vollständigen Eingabe des Pascalprogrammes wird der Editor verlassen. Das editierte Programm wird als Datei abgelegt.

Der Compiler

Nun kann durch ein Batch-File der Compiler aufgerufen werden, der bei fehlerfreiem Quelltext eine Objektkode-Datei erzeugt. Der Compiler kann aber auch unter TOS Anwendung direkt aufgerufen werden.

Es sind auch einige nützliche Compileroptionen verfügbar, die die Programmerstellung vereinfachen.

Es können aber auch Compileranweisungen direkt im Quelltext eingebettet sein. Sie werden durch ein nachgestelltes + oder – an- bzw. ausgeschaltet.

Es existiert eine Auflistanweisung, die den Quellcode samt relativen Adressen auf dem Bildschirm ausgibt.

Eine Einfügeanweisung, die es erlaubt, fremde Dateien in den Quellcode einzufügen. Eine Modulanweisung, die es ermöglicht Programmteile modular zu compilieren.

Eine Debuganweisung sowie einige andere brauchbare Anweisungen, wie etwa Kontrolle auf Bereichsüberschreitung. Probleme gab es jedoch dabei, den Quellcode samt Compilerfehlermeldungen auf ein bestimmtes Ausgabegerät zu lenken. Der Compiler zeigt durch die LIST-Option zwar jede bearbeitete Zeile samt eventuell vorhandenen Fehlermeldungen auf dem Monitor, jedoch war es nicht möglich, in der kurzen Zeit, in der mit dem Compiler gearbeitet wurde, diese Ausgabe an einen Drucker oder eine Datei umzuleiten.

Dies ist vor allem bei der Erstellung komplexerer Programme ein großes Handicap, da der compilierte Text doch recht schnell über den Bildschirm huscht und die darin enthaltenen Fehlermeldungen für den normalen Menschen nicht gerade ersichtlich werden.

Der Compiler erzeugt ca. 150 verschiedene Fehlermeldungen in englischem Klartext, was heute eigentlich schon Standard ist. In ST Pascal ist das komplette Standardpascal nach Wirth und Jensen implementiert, das um die OTHERWISE- und LOOP-Anweisung erweitert wurde.

Des weiteren unterstützt es Random-Access Dateien, erleichtert die Handhabung von Dateien im allgemeinen und ermöglicht mit Einschränkungen die Darstellung doppelt langer Integers. GOTOs sind nur innerhalb von Blöcken erlaubt.

Mit CHAIN kann von Pascal aus in ein fremdes Programm übergewechselt werden und mit EXTERNAL können externe Funktionen oder Prozeduren aufgerufen werden. Außerdem können Funktionen und Prozeduren als Parameter übergeben werden, was bei den meisten anderen heute verfügbaren Pascalsystemen nicht möglich ist. Zusätzlich wurde fast das gesamte UCSD-Pascal implementiert, was vor allem bei der Stringverarbeitung von unschätzbarem Vorteil ist. Es fehlen die UNIT-Anweisungen ganz, was jedoch absolut nicht von Nachteil ist, da sie sowieso nur bei sehr wenigen Systemen vorkommen und die bei ST Pascal mögliche modulare Compilierung diese Anweisungen voll und ganz ersetzt.

Ein großer Vorteil von ST Pascal ist die Möglichkeit Prozeduren und Funktionen, die in Assembler oder C geschrieben sind, zu integrieren.

Um etwa die VDI-Funktion 'set fill color index', also setzen der Füllfarbe, aufzurufen, ist in ST Pascal folgende Befehlsfolge nötig: PROCEDURE vsf color(handle,color index: integer); C;

Um AES-Routinen aufzurufen, wird genauso verfahren. Weiterhin ist es möglich, GEMDOS- oder BIOS-Funktionen direkt von ST Pascal aus zu programmieren.

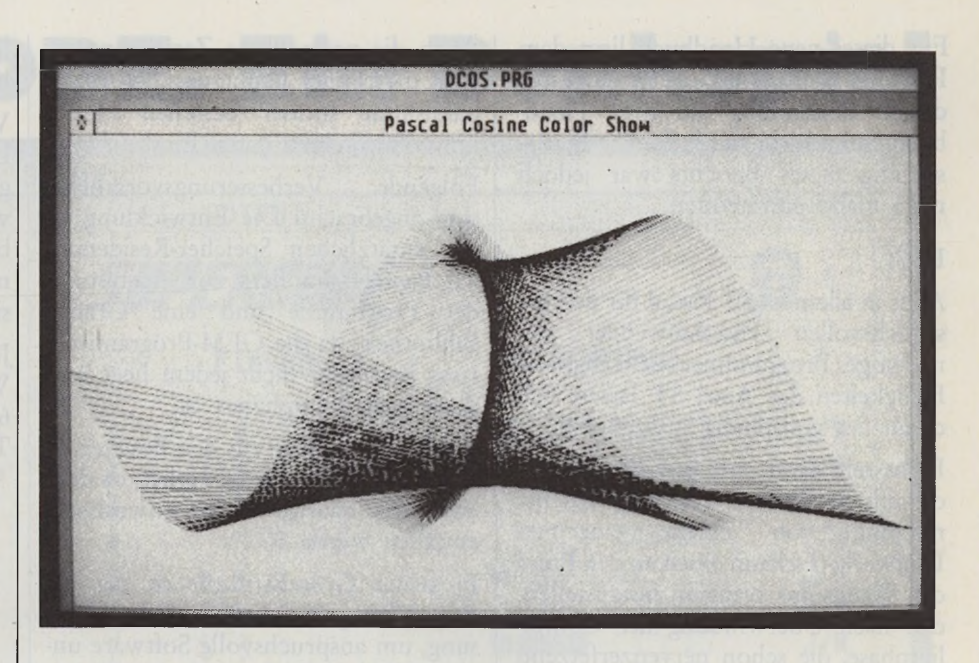
Um den Rechner z. B. in den Warmstart zu versetzen, genügt die Befehlsfolge:

```
FUNCTION warm;
GEMDOS(0);
```

ST Pascal ist so konzipiert, das auf dem Atari ST erstellten Programme auf jeden 68 000-Rechner unter GEM lauffähig sind. Mit diesen Fähigkeiten ausgestattet, ist es eine wahre Freude, mit ST Pascal zu programmieren.

Die Compilergeschwindigkeit des ST Pascal liegt bei etwa 260 Zeilen pro minute, was zwar nicht sehr beeindruckend ist, aber sich beim Arbeiten als erträglich erwies.

Der von dem Compiler erzeugte Objektcode hat im allgemeinen eine bis zu doppelte Größe wie der entsprechende Sourcekode.



Der Linker

Nun muß bei nur einem Laufwerk die Objektdatei von der Compilerdiskette auf die Linkerdiskette kopiert werden. Danach wird mit 'Li' oder 'LGM1' der Linker aktiviert.

Bei zwei Laufwerken wird mit dem Batch-Aufruf 'CL2' bzw. 'CLGM2' das Quellprogramm in einem Durchgang compiliert und gelinkt.

Der Linker, der übrigens von Digital Research stammt, bindet die benötigten Objektmodule und GEM-Anwendungen aus den mitgelieferten VDI- und AES-Bibliotheken in das Objektprogramm ein und generiert ein ausführbares Programm. Das Linken kann dabei ein vielfaches der Compilierzeit erreichen.

Zusätzlich werden noch zwei Pascal-Programme im Sourcetext mitgeliefert, an denen sich der Anwender mit compilieren und linken vertraut machen kann.

Vor allem das Programm CDOS demonstriert sehr schön die Anwendung von GEM-Routinen unter Pascal.

CDOS erzeugt eine Grafik auf dem Bildschirm, die zwar weder besonders schön noch besonders schnell ist, aber die grundlegenden Möglichkeiten von ST Pascal ersichtlich macht.

Die Compilerzeit von CDOS betrug ca. 57 Sekunden und das Linken benötigte nochmal ca. 2 Minuten und 49 Sekunden.

Das Quellprogramm hatte einen Umfang von 252 Zeilen und benötigte einen Speicherplatz von 5 673 Bytes.

Als prg.-File belegte es 9 128 Bytes an Speicher und ist direkt, also ohne irgend ein 'run-time-module', Interpreter oder ähnlichem, ausführbar.

Einige kleine Benchmarks ergaben einen Geschwindigkeitsvorteil gegenüber dem ST Basic von nur dem vier- bis fünffachen und bei dem Speicherbedarf verlangte ST Pascal gleich einen bis zu 40 mal größeren Bereich.

Wobei ST Pascal eine sehr viel höhere Rechengenauigkeit angerechnet werden muß und kein kodeverkürzender Interpreter benötigt wird. Von CCD ist zu dem Problem der Geschwindigkeit und des recht großen Speicherplatzbedarfs schon eine Lösung angekündigt: Der Compiler soll demnächst um die Option 'Optimize' erweitert werden, wordurch effizienterer Objektkode erzeugt werden soll.

Ferner ist ein besseres Handbuch angekündigt, das näher auf die Eigenschaften von ST Pascal auf dem Atari ST eingehen wird. Das vorliegende Handbuch bezieht sich noch im wesentlichen auf das CP/M-68K-Pascal, worauf das ST Pascal im großen ganzen aufbaut.

Für dieses neue Handbuch liegt dem Programmpaket ein Gutschein bei, bei dessen Einsendung das neue Handbuch zugeschickt wird. Bei der Fertigstellung dieses Berichts war jedoch noch nichts eingetroffen.

Fazit

Alles in allem ist ST Pascal für den anspruchsvollen Pascalanwender ein mächtiges Programmierwerkzeug. Die Fähigkeiten des Atari ST lassen sich damit fast vollständig ausschöpfen.

Es besteht zwar eine gewisse Anwenderunfreundlichkeit vor allem bei Benutzung von einem einzelnen Laufwerk. (Es kann einen an den Rand des Wahnsinns bringen, festzustellen, daß nach Überwindung der Compilierphase, die schon nervenzerfetzend genug ist, und nach kopieren des Objektprogrammes und anschließendem Linken immer noch etwas nicht stimmt.)

Auch die permanente Zerstörungsgefahr, die bei der Benutzung der Originaldiskette immer bestehen bleibt, hinterläßt keinen guten Eindruck.

Folgende Verbesserungsvorschläge sind angebracht: Die Entwicklung eines zusätzlichen Speicher-Residenten ST Pascal-Compilers zur Austestung der Programme und eine Grafik-Bibliothek, da die GEM-Programmierung bestimmt nicht jedem liegt und doch recht kompliziert ist.

Auch eine Version für den Besitzer eines doppelseitigen Laufwerkes, da dort auch das häufige Diskettenwechseln entfallen würde.

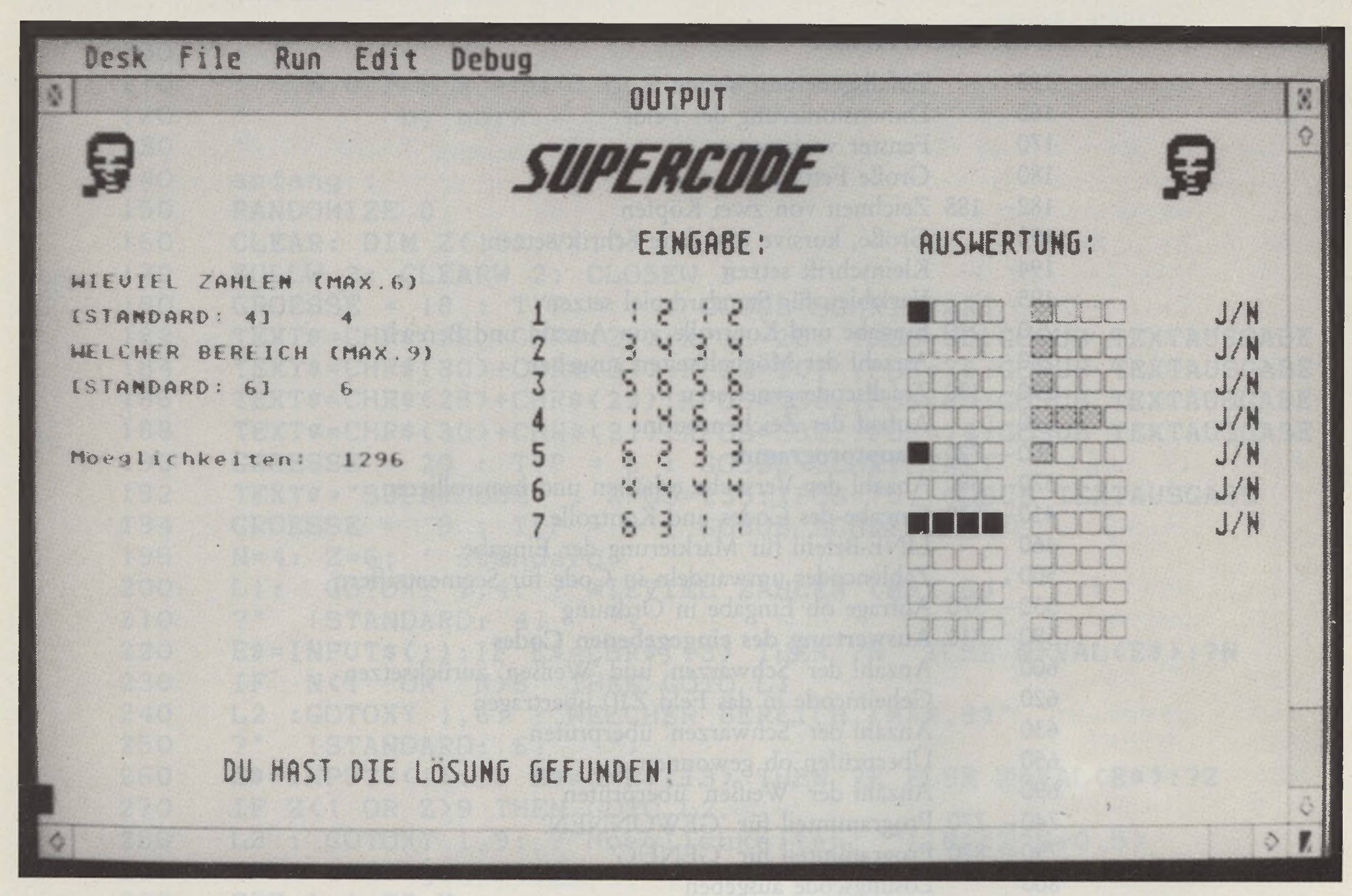
In seiner Grundstruktur ist der ST Pascal-Compiler zur Zeit die beste Lösung, um anspruchsvolle Software unter Pascal auf dem Atari ST zu entwickeln.

Einige Einschränkungen werden wohl hoffentlich bei der nächsten Version beseitigt sein.

Vor allem für den Besitzer von zwei Laufwerken ist ST Pascal zu seinem gegenwärtigen Preis von 249 DM wirklich unübertroffen. Abzuwarten bleibt jedoch, wie gut Borland darauf mit einer auf den ST angepaßten Version von Turbo-Pascal kontern kann...

Jörg Mainusch
Wolfsgartenstr. 70
6070 Langen
Tel. 06103/27166

SUPERCODE: Ein Denkspiel



PROGRAMMBESCHREIBUNG

Bei diesem Denkspiel geht es darum einen Geheimcode, den der Computer erzeugt, zu erraten. Jeder Versuch wird dabei bewertet, das heißt, daß angegeben wird wieviel Zahlen mit denen des Geheimcodes in Wert und Stelle übereinstimmen. Das Programm malt dann die entsprechende Anzahl von Kästchen 'SCHWARZ' aus. Wenn die Stelle der Zahl mit der des Geheimcodes nicht übereinstimmt, die Zahl aber an einer anderen Stelle vorhanden ist, so wird dies mit 'WEISSEN' Kästchen angezeigt. Wenn Sie Ihren Code eingegeben haben, fragt das Programm, ob Sie diesen Code nehmen wollen. Sollten Sie diese Frage mit N bzw. n beantworten, dann bleibt zwar der 'alte' Code auf dem Bildschirm stehen, doch sie können bzw. müssen nun erneut einen Code eintippen. Für das Erraten des Geheimcodes haben Sie zehn Versuche zur Verfügung, wenn Sie den Code bis dahin erraten haben, bekommen Sie dafür einen Tusch gespielt. Sollten Sie den Code mit dem zehnten Versuch nicht erraten haben hört das Programm auf und gibt einen Text und den Code aus. Außerdem können Sie die Runde jederzeit abbrechen wenn Sie keine Lust mehr haben. Drücken Sie dafür einfach die Taste 'E'. Die Anzahl der Versuche mag zwar gering erscheinen, doch bei genauer Analyse der 'Schwarzen' und der 'Weißen' erhält man genug Zusatzinformationen, um den Code schnell zu bestimmen.

Beim Starten des Programms (wobei man immer im 'EDIT'-Modus sein sollte!), wird zuerst nach der Anzahl der Zahlen des Geheimcodes gefragt. Man kann hier bis zu sechs Zahlen wählen. Damit läßt sich bei einem entsprechenden Zahlenbereich ein hoher Schwierigkeitsbereich erreichen (über 300 000 Möglichkeiten). Als nächstes erwartet das Programm die Eingabe des Bereichs, der von 1 bis 9 gewählt werden kann. Bei beiden Eingaben kann man durch Drücken der <Return>-Taste die Standardwerte einstellen. Nachdem die Anzahl der Möglichkeiten angezeigt wurde, können Sie Ihren Zahlencode eingeben. Die einzelnen Ziffern werden, um sie besser hervorzuheben, als 'Segmentziffern' ausgegeben, wie sie auch bei Taschenrechnern und Digitaluhren verwendet werden.

Zum Einschalten der verschiedenen, im Normalbasic nicht ansprechbaren Schriftarten, werden die GEM-VDI Routinen aufgerufen. Nähere Erklärungen dazu finden Sie im GEM-Artikel in diesem Heft.

Beim Eintippen des Programms muß auf die Zeilennummern keine Rücksicht genommen werden, da das Programm nur Sprünge zu 'Labels' enthält.

VARIABLENLISTE

CODE$	Code
Z()	Code (indizierte Variable)
Z1()	Code für Vergleich
V()	Versuch
N	Anzahl der Zahlen
Z	Bereich der Ziffern
E$	allgemeine Eingabevariable
A,B	allgemeine Schleifenvariablen
VN	Anzahl der Versuche
S	Anzahl der 'Schwarzen'
W	Anzahl der 'Weißen'
X,Y	Variablen für LINE und FILL

HAUPTROUTINEN

Zeilen	Funktion
150	Zufallsgenerator setzen
160	Dimensionierung der Felder
170	Fenster vorbereiten
180	Große Fettschrift setzen
182 – 188	Zeichnen von zwei Köpfen
190	Große, kursive und fette Schrift setzen
194	Kleinschrift setzen
195	Variablen für Standardspiel setzen
200 – 270	Eingabe und Kontrolle von Anzahl und Bereich
280	Anzahl der Möglichkeiten ausgeben
290 – 330	Zufallscode generieren
340	Aufruf der Zeichenroutine
360 – 720	**Hauptprogramm**
370 – 380	Anzahl der Versuche erhöhen und kontrollieren
410 – 570	Eingabe des Codes und Kontrolle
460	LINE-Befehl für Markierung der Eingabe
500	Zahlencodes umwandeln in Code für Segmentziffern
530 – 570	Abfrage ob Eingabe in Ordnung
590 – 710	**Auswertung des eingegebenen Codes**
600	Anzahl der 'Schwarzen' und 'Weißen' zurücksetzen
620	Geheimcode in das Feld Z1() übertragen
630	Anzahl der 'Schwarzen' überprüfen
650	Überprüfen ob gewonnen
690	Anzahl der 'Weißen' überprüfen
740 – 770	Programmteil für 'GEWONNEN'
790 – 820	Programmteil für 'GENUG'
800	Lösungscode ausgeben
840 – 870	Programmteil für 'AUFGEGEBEN'
880	Programmende
900 – 1040	**Unterprogramm zum Zeichnen der Vierecke**
905	Standardschrift setzen
930	y-Koordinate berechnen
950	Lücke zwischen 'Schwarzen' und 'Weißen'
960	y-Koordinate berechnen
970 – 1000	LINE-Befehl für Viereck
1060 – 1110	**Unterprogramm zum Ausfüllen der 'Schwarzen'**
1070 – 1080	x- und y-Koordinaten berechnen
1090 – 1100	Farbe setzen und ausfüllen
1130 – 1180	**Unterprogramm zum Ausfüllen der 'Weißen'**
1140 – 1150	x- und y-Koordinaten berechnen
1160 – 1180	COLOR setzen und ausfüllen
1200 – 1310	**Unterprogramm 'Tusch' für 'Gewonnen'**
1210	Ausschalten der drei Tonkanäle
1220 – 1240	Akkord mit drei Tonkanälen setzen
1250	Tonkanäle einschalten und Ton halten
1260 – 1290	Zweiten Akkord setzen und spielen
1400 – 1500	**Unterprogramm zum Setzen von Schriftgröße und Schriftart GEM-Routinen 107,106**
1510 – 1630	**Unterprogramm zur maßstabsgerechten Textausgabe GEM-Routine 8**

```
List of \CODE99.BAS

 100   '
 110   '    S U P E R - C O D E
 120   '         by mark
 130   '
 140   anfang :
 150   RANDOMIZE 0
 160   CLEAR: DIM Z(10),V(10)
 170   FULLW 2: CLEARW 2: CLOSEW 3
 180   GROESSE = 18 : TYP = 1 : GOSUB SCHRIFTART
 182   TEXT$=CHR$(28)+CHR$(29):XPOS=20: YPOS=60:GOSUB TEXTAUSGABE
 184   TEXT$=CHR$(30)+CHR$(31):XPOS=20: YPOS=76:GOSUB TEXTAUSGABE
 186   TEXT$=CHR$(28)+CHR$(29):XPOS=550:YPOS=60:GOSUB TEXTAUSGABE
 188   TEXT$=CHR$(30)+CHR$(31):XPOS=550:YPOS=76:GOSUB TEXTAUSGABE
 190   GROESSE = 20 : TYP = 5 : GOSUB SCHRIFTART
 192   TEXT$="SUPERCODE" : XPOS=235:YPOS=75: GOSUB TEXTAUSGABE
 194   GROESSE =  9 : TYP = 0 : GOSUB SCHRIFTART
 195   N=4: Z=6: ' standard
 200   L1: GOTOXY 1,4: ?"WIEVIEL ZAHLEN (MAX.6)"
 210   ?"  [STANDARD: 4] ";
 220   E$=INPUT$(1):IF E$=CHR$(13) THEN ?N  ELSE N=VAL(E$):?N
 230   IF  N<1  OR  N>6  THEN GOTO L1
 240   L2 :GOTOXY 1,6: ?"WELCHER BEREICH (MAX.9)"
 250   ?"  [STANDARD: 6] ";
 260   E$=INPUT$(1):IF E$=CHR$(13) THEN ?Z ELSE Z=VAL(E$):?Z
 270   IF Z<1 OR Z>9 THEN GOTO L2
 280   L3 : GOTOXY 1,9: ?"Moeglichkeiten: ";INT(Z^N+0.5)
 290   '    code generieren
 300   FOR A=1 TO N
 310   Z(A)=INT(RND(1)*Z)+1
 320   CODE$=CODE$+STR$(Z(A))
 330   NEXT A
 340   GOSUB einleitung
 350   '-------------------------------------------------------
 360   hauptprogramm :
 370   VN=VN+1
 380   IF VN>10 THEN GOTO genug
 390   GOTOXY 14,4+VN: ?VN
 400   '-------------------------------------------------------
 410   codeeingabe :
 420   FOR A=1 TO N
 430   GOTOXY 34,4+VN: ?"   "
 440   '-------------------------------------------------------
 450   eingabe :
 460   LINEF 272+17*A,(5+VN)*17,279+17*A,(5+VN)*17
 470   E$=INPUT$(1): IF LEN(E$)<>1 THEN GOTO eingabe
 480   IF E$="E" OR E$="e" THEN GOTO abbruch ELSE V(A)=VAL(E$)
 490   IF V(A)<1 OR V(A)>Z THEN GOTO eingabe
 500   GOTOXY 16+A,4+VN: ? chr$(v(a)+16)
 510   NEXT A
 520   '-------------------------------------------------------
 530   annahme :
 540   GOTOXY 34,4+VN: ?"J/N"
```

List of \CODE99.BAS

```
550   E$=INPUT$(1): IF E$=CHR$(13) THEN GOTO auswertung
560   IF E$="J" OR E$="j" THEN GOTO auswertung
570   IF E$="N" OR E$="n" THEN GOTO codeeingabe ELSE GOTO annahme
580   '------------------------------------------------------------
590   auswertung :
600   S=0: W=0
610   FOR A=1 TO N
620   Z1(A)=Z(A)
630   IF Z1(A)=V(A) THEN S=S+1: V(A)=0: Z1(A)=0: GOSUB schwarz
640   NEXT A
650   IF S=N THEN GOTO gewonnen
660   FOR A=1 TO N
670   IF V(A)=0 THEN GOTO belegt
680   FOR B=1 TO N
690   IF V(A)=Z1(B) THEN W=W+1:Z1(B)=0:V(A)=0:GOSUB weiss:GOTO belegt
700   NEXT B
710   belegt : NEXT A
720   GOTO hauptprogramm
730   '------------------------------------------------------------
740   gewonnen :
750   GOTOXY 5,18: ?" DU HAST DIE LöSUNG GEFUNDEN!
760   GOSUB tusch
770   E$=INPUT$(1): GOTO anfang
780   '------------------------------------------------------------
790   genug :
800   GOTOXY 1,12: ?"("CODE$" )"
810   GOTOXY 5,18: ?"  Das war wohl nichts, SCHAD"chr$(144)chr$(144)
820   E$=INPUT$(1): GOTO anfang
830   '------------------------------------------------------------
840   abbruch :
850   GOTOXY 1,12: ?"("CODE$" )"
860   GOTOXY 5,18:?"WARUM HAST DU NACH"VN-1"VERSUCHEN AUFGEGEBEN ?"
870   E$=INPUT$(1): GOTO anfang
880   end
890   '------------------------------------------------------------
900   einleitung :
905   GROESSE=10: TYP=0: GOSUB SCHRIFTART
910   GOTOXY 17,3: ?" EINGABE:          AUSWERTUNG:"
920   FOR A=1 TO 10
930   Y=A*17+73
940   FOR B=1 TO 2*N+1
950   IF B=N+1 THEN GOTO luecke
960   X=440-N*6+B*12
970   LINEF X,Y,X+10,Y
980   LINEF X+10,Y,X+10,Y+10
990   LINEF X+10,Y+10,X,Y+10
1000  LINEF X,Y+10,X,Y
1010  luecke :
1020  NEXT B
1030  NEXT A
```

```
List of \CODE99.BAS

1040  RETURN
1050  '-----------------------------------
1060  schwarz :
1070  X=445-N*6+S*12
1080  Y=VN*17+80
1090  COLOR 1,1,1,1,1
1100  FILL X,Y
1110  RETURN
1120  '-----------------------------------
1130  weiss :
1140  X=445-N*6+((W+N+1)*12)
1150  Y=VN*17+80
1160  COLOR 1,1,1,3,2
1170  FILL X,Y
1180  RETURN
1190  '-----------------------------------
1200  tusch :
1210  FOR N=1 TO 3: WAVE 0
1220  SOUND 1,15,1,4,0
1230  SOUND 2,15,8,4,0
1240  SOUND 3,15,1,5,0
1250  WAVE 7: FOR A=0 TO  500: NEXT A: WAVE 0
1260  SOUND 1,15,6,4,0
1270  SOUND 2,15,1,5,0
1280  SOUND 3,15,6,5,0
1290  WAVE 7: FOR A=0 TO 1000: NEXT A
1300  NEXT N: WAVE 0
1310  RETURN
1400  schriftart:
1410  ' --> groesse ; typ
1420  POKE CONTRL,107
1430  POKE CONTRL+2,0
1440  POKE CONTRL+6,1
1450  POKE INTIN,GROESSE
1460  VDISYS
1470  POKE CONTRL,106
1480  POKE INTIN,TYP
1490  VDISYS
1500  RETURN
1510  textausgabe:
1520  ' TEXT$
1530  FOR I=0 TO LEN(TEXT$)-1
1540  POKE INTIN+i*2,ASC(MID$(TEXT$,i+1,1))
1550  NEXT
1560  POKE INTIN+(i   )*2,0
1570  POKE CONTRL,8
1580  POKE CONTRL+2,1
1590  POKE CONTRL+6,LEN(TEXT$)+1
1600  POKE PTSIN , XPOS
1610  poke PTSIN+2,YPOS
1620  VDISYS
1630  RETURN
```

Monitor „scharf" gemacht

Der ATARI Monitor SM 124 ist in der Bildqualität fast jedem anderen, gewöhnlichen Monitor weit überlegen. Diese Tatsache ist jedem bekannt, der einmal mit ihm „gearbeitet" hat. Doch leider hat so mancher dieser Monitore, die übrigens von der Firma GOLDSTAR für ATARI hergestellt werden, seine Schattenseiten. Zumindest scheint eine Endkontrolle bzw. der Abgleich der Geräte öfters nicht mit der nötigen Sorgfalt ausgeführt zu werden. Wie wir selber, und auch bereits von einigen Lesern, erfahren haben, sind folgende Merkmale typisch für einen ungenauen Abgleich der Monitore. Häufig ist eine gewisse Unschärfe des Bildes in der oberen Bildschirmzeile (Desktop) zu erkennen, außerdem scheint eine Aufhellung (im Extremfall sogar ein dünner heller Strich) am linken oder rechten Bildschirmrand nicht selten zu sein. Ein schiefes oder verzerrtes Bild ist dagegen selten vorzufinden. Glücklicherweise bietet der SM 124 im Innern, auf seiner modern konzipierten Platine, alle notwendigen Einstellregler, um eventuelle „Fehler" zu korrigieren.

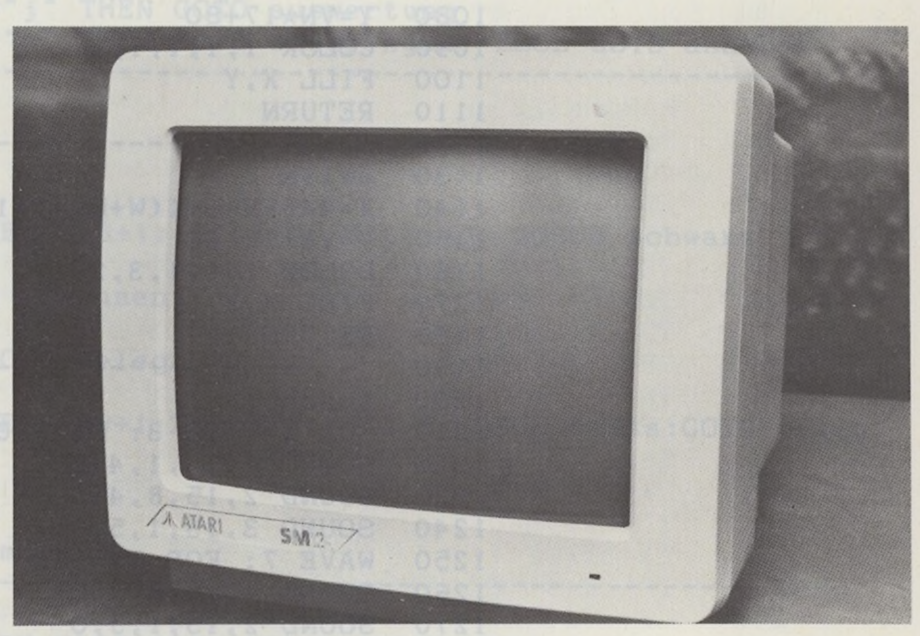
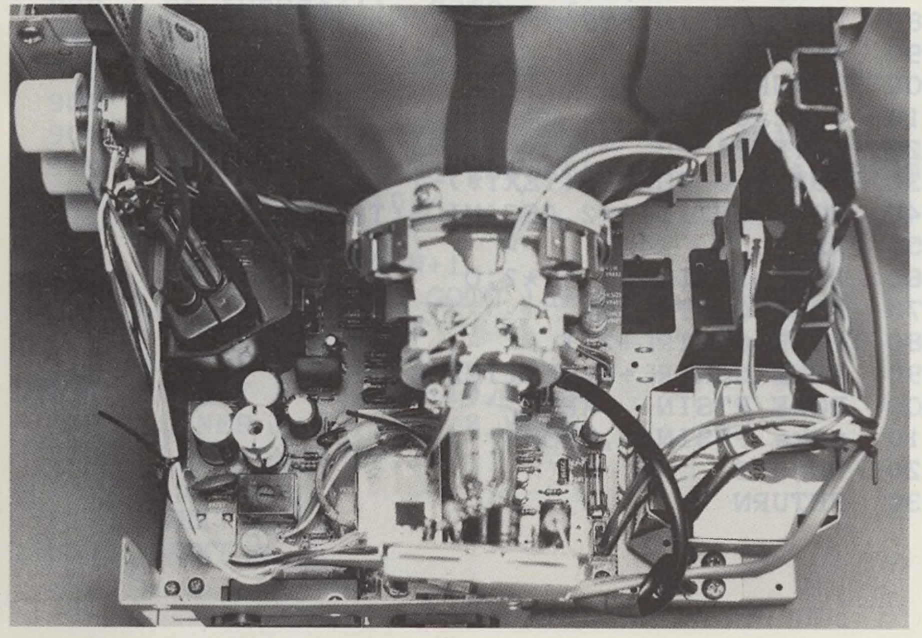


Bild 1: Der monochrome Monitor SM 124

Eine andere Tatsache, die uns von Anfang an störte, ist das zu kleine Bild oder anders gesagt, der zu große schwarze Rahmen. Obwohl in der „Bedienungsanleitung" des Monitors die Bildschirmdiagonale mit 12-inch, das sind über 30 cm, angegeben wird (dieser Wert ist auch zweifellos richtig), werden vom Bild nur ca. 24 cm Bilddiagonale genutzt. Wie man nun seine Bildfläche um ca. 25 %! vergrößern kann und was es dabei zu beachten gibt, ist ein paar Zeilen später genau erklärt.


Bild 2: Das Innenleben des Monitors

Bevor Sie nun gleich zum Schraubendreher greifen, wollen wir nicht verschweigen, daß durch das Öffnen des Gerätes eine Garantie eventuell verfallen kann. Außerdem ist eine korrekte Justierung nur während des Betriebes möglich, so daß die Gefahr eines Stromschlages oder Kurzschlusses bei unsachgemäßen Arbeiten gegeben ist. An dieser Stelle sei auch bemerkt, daß nicht jede Störung des Bildes durch den Monitor verursacht wird. So sind z. B. ein „verschwimmen" des Bildes für eine gewisse Zeit oder einzelne Punkte auf dem Bild typische Fehler des Rechners. Bei Unklarheiten nicht gleich den Monitor aufschrauben und „wild" an allen Reglern drehen, sondern erst einen zweiten Monitor ausleihen (oder zum Fachhändler gehen) und prüfen, ob der Fehler immer noch auftritt, falls ja, ist mit großer Wahrscheinlichkeit Ihr Rechner defekt.

Das Öffnen des Monitors

Vor dem Öffnen ist der Monitor unbedingt vom Netz zu trennen und der

Monitorstecker vom Rechner abzuziehen. Nun ist es am Besten, ihn mit der Bildröhre (also mit der Front) auf die Oberschenkel zu legen, so daß man bequem die fünf Schrauben auf der Rückwand lösen kann. Zwei dieser Kreuzschlitzschrauben befinden sich am oberen Rand, eine weitere direkt über der Netzleitung und nochmals zwei, etwas längere Schrauben, am Gehäuseboden. Das gesamte Rückteil des Gehäuses kann jetzt nach hinten abgezogen werden. Doch Vorsicht, da sich der Lautsprecher im Rückteil befindet müssen noch dessen zwei Leitungen, durch bloßes ziehen, vom Platinenstecker gelöst werden. Der Monitor- und der Netzstecker passen gerade durch die zwei rechteckigen Öffnungen an der Gehäuserückwand, so daß das Rückteil vollständig abgenommen werden kann. Der freigelegte Monitor wird nun, durch anpacken an der Frontseite, auf einen Tisch gestellt.

Das Werkzeug

Wie überall im Leben, so geht auch hier mit dem richtigen Werkzeug alles ganz einfach. Am Besten ist ein sogenanntes Abgleichbesteck geeignet. Dies sind meist drei oder vier Schraubendreher bzw. Imbusschlüssel verschiedener Größen aus **Plastik** in einem Set. Ein solches Abgleichset sollte man im guten Elektronikfachhandel erwerben können. Da der Elektronenstrahl einer Bildröhre durch ein Magnetfeld, welches durch Spulen auf dem Röhrenhals erzeugt wird, gesteuert bzw. abgelenkt wird, genügt schon ein Metallschraubendreher in der Nähe der Bildröhre, um das Bild zu verzerren. Ein weiterer Vorteil von Plastik-Werkzeug ist die Tatsache, daß man die Ferrit-Kerne von Spulen (siehe H. Size und H. Line) problemlos verdrehen kann. Ein Metallschraubendreher verändert schon bei Berührung des Ferrit-Kernes die Induktivität (den Wert) der Spule, so daß das Bild verändert wird, ohne das der Kern der Spule verstellt wurde. Selbstverständlich geht das Bild in seine normale „Darstellung" zurück, sobald der Schraubendreher wieder entfernt wird. Außerdem verhindert Plastik-Werkzeug die Gefahr eines versehentlichen Kurzschlusses oder eines Stromschlages bei Berührung spannungsführender Leitungen, denn immerhin ent-

```
List of \TESTBILD.BAS

10      '                       TESTBILD     V1.0
20      clear
30      clearw 2: fullw 2
40      color 1,1,1,3,3
50      circle 308,172,150
60      fill 308,172
70      x=5  :y= 5
80      color 1,1,1,2,3
90      gosub rechteck
100     x=5  :y=180
110     color 1,1,1,3,3
120     gosub rechteck
130     x=570:y=5
140     color 1,1,1,4,3
150     gosub rechteck
160     x=570:y=180
170     color 1,1,1,5,3
180     gosub rechteck
190     color 1,1,1,6,2
200     x1=115 :y1=60
210     gosub kreis
220     color 1,1,1,7,2
230     x1=115 :y1=284
240     gosub kreis
250     color 1,1,1,8,2
260     x1=501 :y1=284
270     gosub kreis
280     color 1,1,1,9,2
290     x1=501 :y1=60
300     gosub kreis
310     art=4:gosub schrift
320     gotoxy 10 ,1  : print "Testbild"
330     gotoxy 23 ,1  : print "Testbild"
340     gotoxy 10 ,18 : print "Testbild"
350     gotoxy 23 ,18 : print "Testbild"
360     color 1,1,1,12,3
370     fill 1,100
380     a=inp (2)
385     art=1:gosub schrift
390     end
400     rechteck :
410     linef x,y,x+40,y
420     linef x,y,x,y+160
430     linef x+40,y,x+40,y+160
440     linef x,y+160,x+40,y+160
450     fill x+10,y+80
460     return
470     kreis :
480     circle x1,y1,50
490     fill x1,y1
500     return
510     schrift :
520     poke contrl,32
530     poke contrl+2,0
540     poke intin,art
550     vdisys
560     return
```

stehen im Monitor Spannungen über 12 000 Volt! Wer kein Abgleichset zur Verfügung hat, der kann mit den oben erwähnten Schwierigkeiten, auch einen gewöhnlichen kleinen Schraubendreher verwenden. Allerdings sollte dieser einen möglichst langen isolierten Stiel haben, um auch etwas „versteckt“ sitzende Einstellregler erreichen zu können.

Bevor Sie den Monitor abgleichen, sollten Sie das kleine Basic Testprogramm eingeben (siehe Foto vom Testbild), damit auch nach dem Abgleich ein CIRCLE-Befehl einen Kreis zeichnet und keine schiefe Ellipse. Haben Sie das Programm eingetippt, dann starten sie es mit RUN und lassen das Testbild während der gesamten Abgleicharbeiten auf dem Monitor stehen. Somit haben Sie ständig eine optische Kontrolle über die Wirkung der einzelnen Regler. Sie sollten sich über Ihr Problem im klaren sein und nur die Trimmpotis oder Spulenkerne (siehe folgende Tabelle) verstellen, die für Sie in Frage kommen. Am sichersten ist es, sich alle Einstellungen zu merken oder aufzuschreiben, dann kann man nichts schlechter machen als es war.

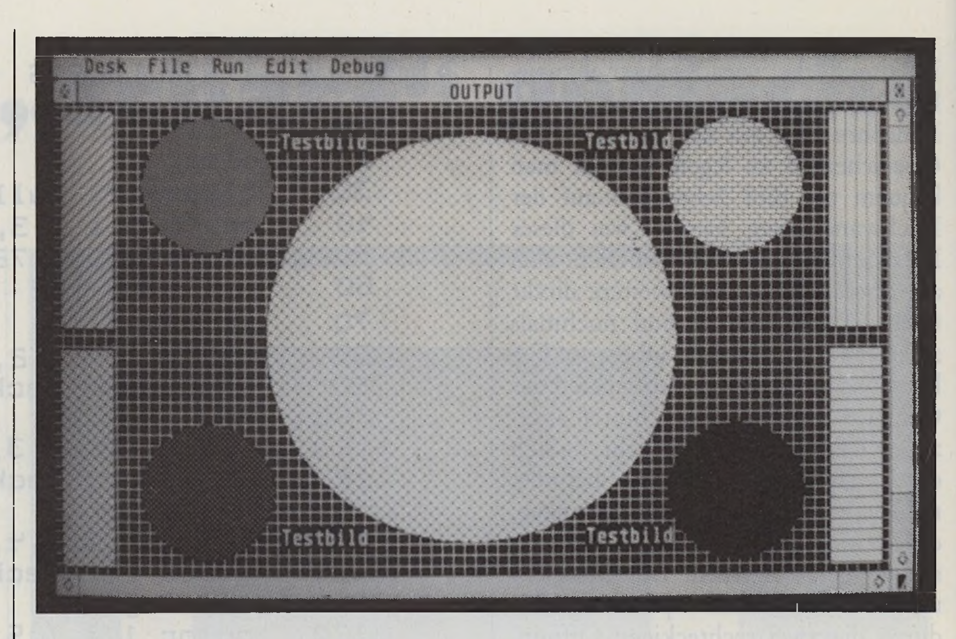


Bild 3: „Ergebnis“ des Basicprogrammes Testbild

Vergrößern des Bildes

Um das Bild zu vergrößern ist es erforderlich den Spulenkern der **H.SIZE** Spule im Uhrzeigersinn reinzudrehen (Imbus 3 mm oder notfalls entsprechender Schraubendreher), ebenfalls ist das Poti **V.SIZE** im Uhrzeigersinn zu verstellen. Damit ein Quadrat nicht zum Rechteck wird, müssen beide Regler im gleichen Verhältnis verdreht werden. Eine optische Kontrolle dafür bietet das Testbild, genauer funktioniert es jedoch durch Ausmessen der Bild-Seitenlängen mit Hilfe eines Roll-Zentimetermaßes. Bei genauem Abgleich muß der Quotient aus horizontaler- und vertikaler Seitenlänge 1,6 ergeben. Dies bedeutet in der Praxis, hat man das Bild in horizontaler Richtung (X-Richtung) auf 23 cm vergrößert, so muß nun das Bild in vertikaler Richtung (Y-Richtung) auf ca. 14,4 cm vergrößert werden (23/14,4 = 1,6).

Tabelle der verschiedenen Einstellmöglichkeiten

Bezeichnung	Bauteil	Wirkung
VR 703 **FOCUS**	Poti	Schärfe im Bildmittelpunkt und im Randbereich
VR 702 **SUB-BRIGHT**	Poti	Grundhelligkeit des Bildes
VR 701 **H.CENTER**	Poti	horizontale Zentrierung des Bildes
L 703 **H.LINE**	Spule mit Kern	horizontale Linearität des Bildes
L 702 **H.SIZE**	Spule mit Kern	horizontale Bildgröße
VR 601 **V.HOLD**	Poti	Bildfang (vertik. Rollen)
VR 602 **V.SIZE**	Poti	vertikale Bildgröße
VR 603 **V.LINE**	Poti	vertikale Linearität des Bildes
siehe Foto (auf Bildröhre)	Ferritring mit Lasche (Zunge) (2 Stück)	diagonales verschieben des Bildes; Krümmung am Rand (vertikale Zentrierung)

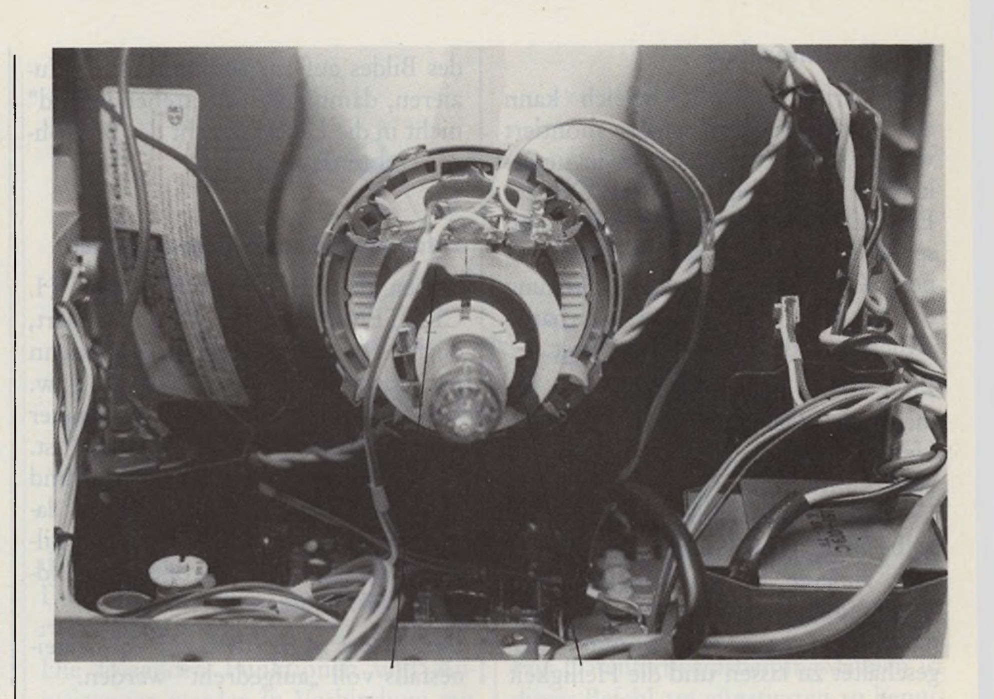
Unscharfes Bild am Rand oder in der Mitte

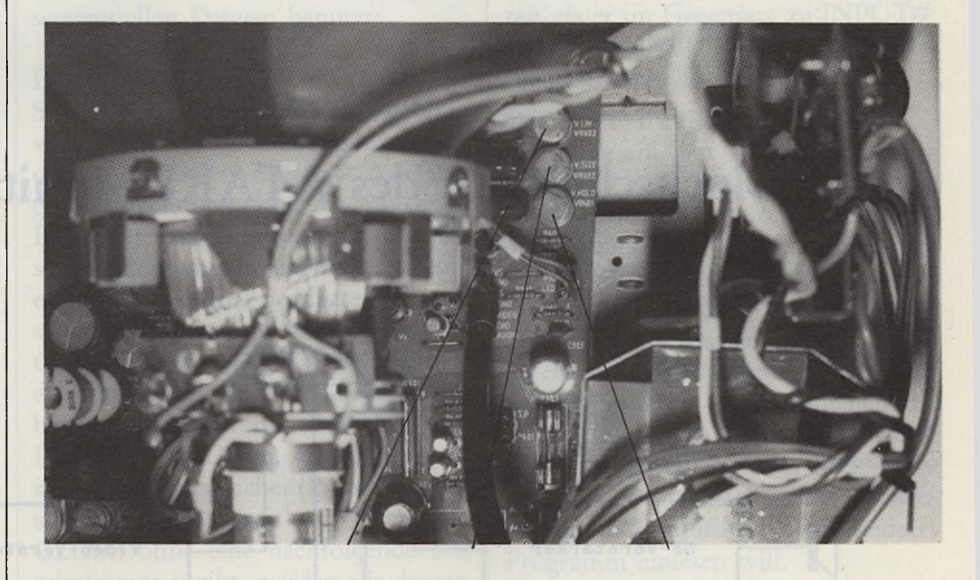
Für die Bildschärfe ist der **FOCUS** Regler zuständig. Durch verdrehen dieses Potis können Sie entweder den Rand oder den Bildmittelpunkt gestochen scharf abbilden, man muß folglich immer einen Kompromiß zwischen Randschärfe und Bildmittelpunkt wählen. Bei korrekter Einstellung ist das Bild über die gesamte Fläche ausreichend scharf.

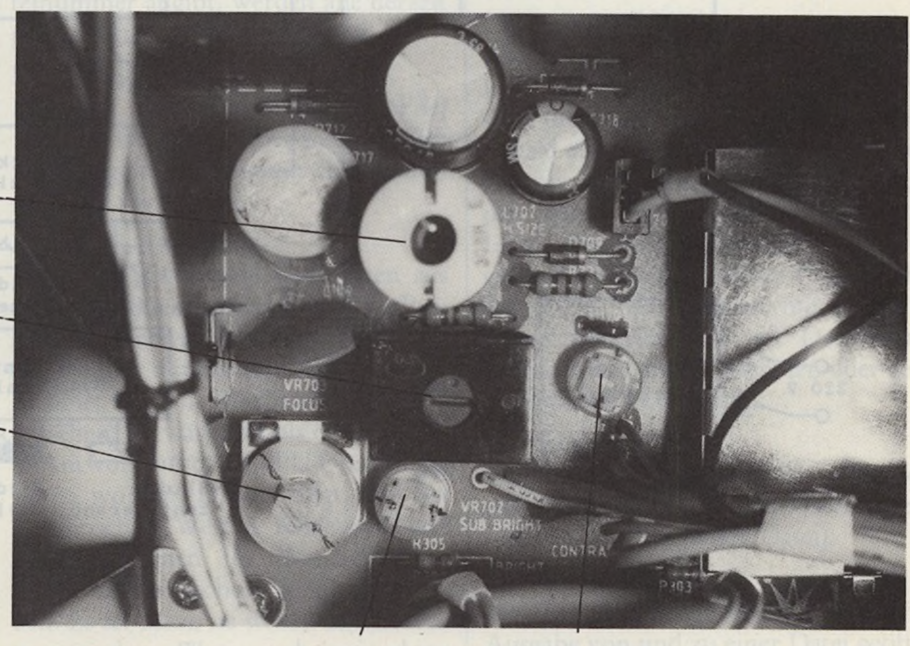
Heller Streifen am rechten oder linken Bildrand

Mit dem Regler **H.CENTER** kann man das Bild in gewissen Grenzen nach rechts bzw. links verschieben. Außerhalb dieser Grenzen entsteht ein heller Streifen am entsprechenden Rand, der schließlich in einen dünnen hellen Strich übergeht. Liegt ein solcher Fehler vor, muß zuerst geprüft werden, ob durch verdrehen dieses Potis der helle Streifen verschwindet. Falls nicht, liegt der Fehler höchstwahrscheinlich nicht am Monitor, sondern am Rechner. Verschwindet der Streifen, so befindet sich Ihr Bild nun sicher nicht mehr in der Mitte. Dies kann nun durch Verstellen der schwarzen Ferritringe auf dem Bildröhrenhals in gewissen Grenzen korrigiert werden. Hierbei kann ein wechselseitiges Verstellen der beiden Ringe und des H.CENTER-Potis erforderlich sein. Sicherheitshalber sollte man sich die Grundstellung der Ferritringe merken.

Die Spule **H.LINE** und das Poti **V.LINE** brauchen meistens nicht korrigiert zu werden.







Der Zusammenbau

Nach erfolgreichem Abgleich kann das Gehäuserückteil wieder montiert werden, dabei verfährt man in umgekehrter Reihenfolge wie beim Öffnen. Es ist darauf zu achten, daß die Lautsprecherbuchse, die wegen ihrer kleinen Kerben nur in einer Stellung richtig paßt, mit einem spürbaren „Klick" auf dem Stecker einrastet.

Zum Schluß noch eine allgemeine Bemerkung. Da die Bildröhre einen Heizfaden, ähnlich einer Glühbirne, besitzt unterliegt sie einem natürlichen Verschleiß. Der Monitor sollte deshalb nicht wegen einer Kaffeepause oder anderen kurzen Unterbrechungen ständig ein- und ausgeschaltet werden. Besser ist es den Monitor eingeschaltet zu lassen und die Helligkeit des Bildes auf ein Minimum zu reduzieren, damit sich das „stehende Bild" nicht in die Leuchtschicht der Bildröhre „einbrennt".

Erfahrungen nach dem Abgleich

Haben Sie das Bild mittels dem H. CENTER-Poti in der Mitte zentriert, dürfen Sie sich nicht wundern, wenn nach dem nächsten Einschalten bzw. Reset Ihr Bild um ein paar Millimeter nach rechts oder links „gewandert" ist. Dieser Effekt ist leider normal und kann nicht behoben werden. Es ist daher ratsam beim „Vergrößern" des Bildes nicht die volle Größe der Bildröhre auszunutzen.

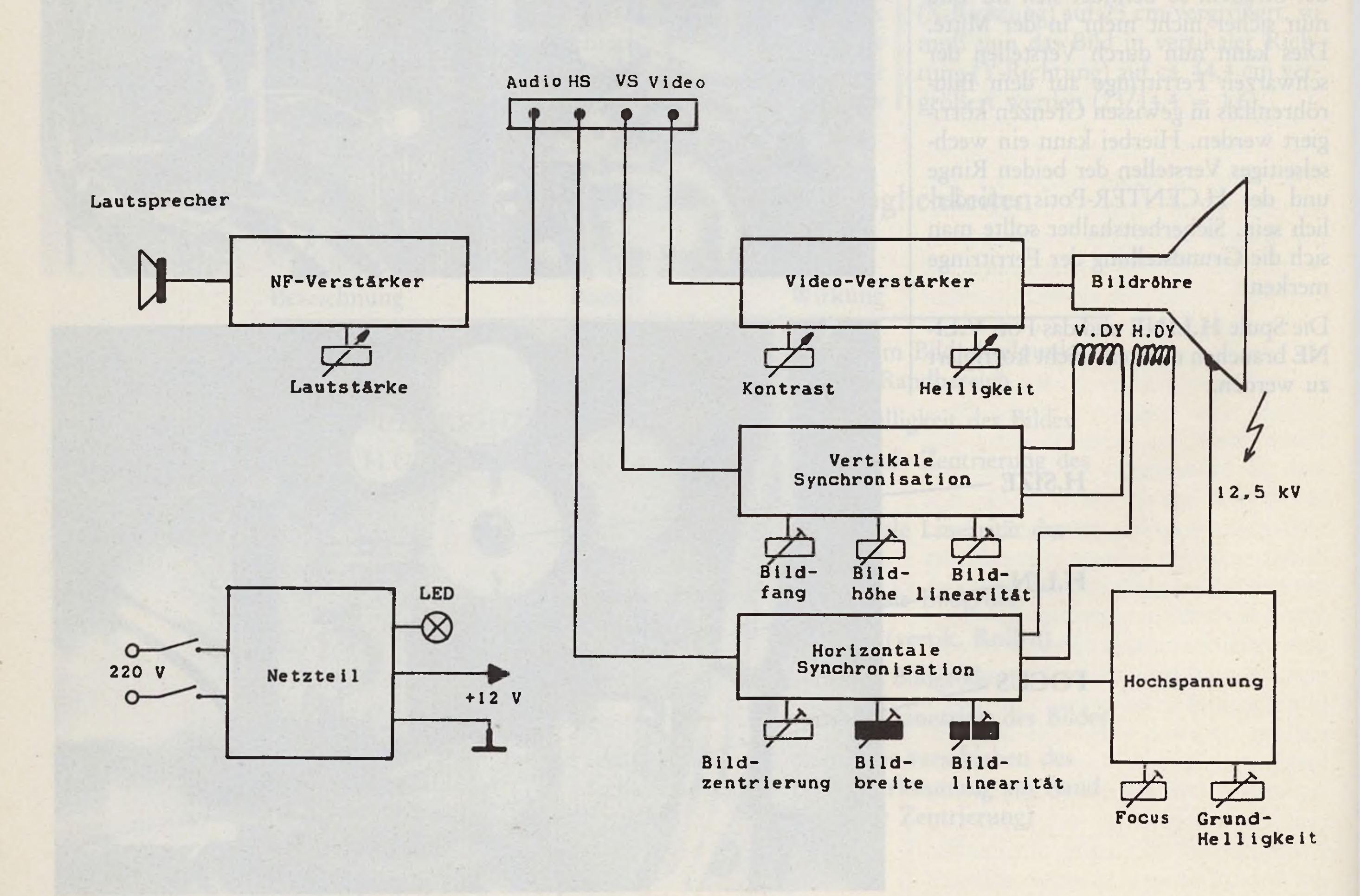
Der Regler SUB BRIGHT sollte keinesfalls voll „aufgedreht" werden.

Technische Daten SM 124

Bildröhre: 30,5 cm diagonal; entspiegelt
Anodenspannung: 15 000 Volt (max.)
Ablenkwinkel: 90 Grad
Beschichtung: Phosphor (papierweiß)
Videobandbreite: ca. 32 MHz
Auflösung: 640 x 400 Pixels; hochauflösend
Horizontale (Zeilen-) Frequenz: 35,7 KHz
Vertikale (Bildwiederhol-) Frequenz: 71 Hz
Arbeitstemperatur: 5 bis 50 Celsius
Netzspannung: 220 Volt – 50 Hz
Leistungsaufnahme: 50 Watt
Gehäuseabmessung: 33 x 31 x 28 cm
Gewicht: 7,7 kg

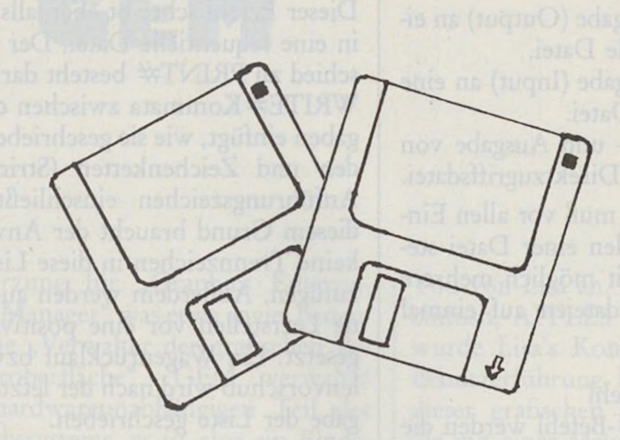
(UB)

Blockschaltbild des ATARI Monitors SM 124



Dateiverwaltung mit

Sequentiellen und Direktzugriffsdateien



In diesem Lehrgang sollen die Grundbegriffe der Dateiverwaltung kurz erklärt und anhand von BASIC-Beispielen aufgezeigt werden. Es werden die Methoden des sequentiellen und des direkten Zugriffs sowie deren Kombination besprochen. Dadurch soll ein Grundstock geschaffen werden, aus dem man dann ohne weiteres eigene Dateien erstellen kann. Natürlich gehören zu einem Dateiverwaltungsprogramm noch mehr als bloße Ein- und Ausleseroutinen. Doch das übrige wie zum Beispiel irgendwelche Sortier- oder Formatierroutinen muß man sich selbst nach eigenen Wünschen gestalten, während die Dateizugriffsmethoden die gleichen bleiben.

Sequentielle Dateien

Sequentielle Dateien haben den Vorteil, daß sie leichter zu erstellen sind als Direktzugriffsdateien, sind aber bei der Verarbeitung der Daten in ihrer Geschwindigkeit und Flexibilität sehr begrenzt. Sie heißt deswegen sequentielle Datei, weil alle Daten nacheinander in ihrer Reihenfolge geschrieben werden. In der gleichen Art und Weise werden sie auch wieder zurückgelesen. Die maximale Menge der Daten ist von der Größe des verfügbaren Arbeitsspeichers abhängig, da man nicht auf einzelne Datensätze auf der Diskette zurückgreifen kann. Somit muß man jedesmal die ganze Datei in vollem Umfang laden, um die Daten zu verarbeiten. Ein Nachteil ist auch, daß man beim ATARI ST BASIC keine Möglichkeit hat, einfach Daten direkt auf Diskette an eine schon bestehende sequentielle Datei anzuhängen. Andere BASIC-Interpreter haben für diesen Fall den APPEND-Befehl. Es bleibt also nur die Möglichkeit seine Datei in den Speicher zu laden, dort die neuen Daten anzuhängen und wieder zurückzuspeichern.

Die folgenden Funktionen und Anweisungen werden in Verbindung mit sequentiellen Dateien benutzt:

Der **CLOSE**-Befehl
Sobald der Befehl CLOSE ausgeführt wird, wird die Beziehung zwischen der Datei und dem Ein- bzw. Ausgabegerät aufgehoben. Eine nachfolgende Ein- oder Ausgabeoperation, die dieselbe Dateinummer anspricht, ist ohne ein erneutes öffnen der Datei mit derselben Dateinummer nicht mehr gültig. Man kann dieselbe Dateinummer dann aber auch für eine andere Datei benutzen. Durch das Schließen der Datei werden Daten, die noch in einem Zwischenspeicher (Puffer) stehen, auf Diskette abgespeichert. Wenn man CLOSE ohne eine nachfolgende Dateinummer angibt, werden alle derzeit geöffneten Dateien geschlossen.

Der **EOF**-Befehl
Mit dem EOF-Befehl (End of File) wird das Dateiende einer sequentiellen Datei abgefragt und somit eine Fehlermeldung vermieden.

Der **INPUT#**-Befehl
Der INPUT#-Befehl liest Daten aus einer Diskettendatei und weist sie Programmvariablen zu. Zuvor muß zuerst die Datei mittels des OPEN-Befehls geöffnet werden. Die Daten müssen so aussehen, als wären sie als eine normale Antwort auf den INPUT-Befehl gegeben worden. Daraus folgt, daß bei numerischen Werten führende Leerstellen, Zeichen für Wagenrücklauf oder Zeilenvorschub ignoriert werden. Es sei denn, sie stehen hinter dem numerischen Wert, wobei sie dann aber nur als Ende des Wertes erkannt werden.

Der **INPUT$**-Befehl
Für Datenfernverarbeitungsdateien ist dieser Befehl am günstigsten zu benutzen, da er im Gegensatz zu INPUT# und LINE INPUT# alle ASCII-Zeichen aus einer Datei einliest.

Der **LINE INPUT#**-Befehl
Dieser Befehl liest alle Zeichen einer Datei bis zum Zeichen für Wagenrücklauf bzw. Zeilenvorschub. Zuvor muß zuerst die Datei mittels des OPEN-Befehls geöffnet werden. Alle Daten einschließlich irgendwelcher Trennzeichen werden dabei in eine Zeichenkette übernommen. Die maximale Anzahl der übernommenen zeichen beträgt 254. Der LINE INPUT#-Befehl ist zum Beispiel zu benutzen, wenn man ein BASIC-Programm in Datenform in ein anderes Programm einlesen will.

Der **LOC**-Befehl
Bei sequentiellen Dateien übergibt LOC die Anzahl der gelesenen Datensätze der Datei seit sie eröffnet wurde. Sobald eine sequentielle Datei für eine Eingabe eröffnet wird, liest das BASIC den ersten Sektor der Datei, so daß LOC schon eine 1 übergibt, bevor irgendeine Eingabe von der Datei gelesen wurde. Bei Dateien mit Direktzugriff übergibt LOC die Datensatznummer des letzten gelesenen oder geschriebenen Datensatzes.

Der **LOF**-Befehl
Dieser Befehl übergibt die aktuelle Anzahl der Bytes, die der Datei zugeordnet sind (Länge der Datei).

Der **OPEN**-Befehl
Mit diesem Befehl wird die Ein- bzw. Ausgabe von und zu einer Datei geöff-

net. Es sind dabei drei Modi möglich:

"O" steht für Ausgabe (Output) an eine sequentielle Datei.

"I" steht für Eingabe (Input) an eine sequentielle Datei.

"R" steht für Ein- und Ausgabe von bzw. an eine Direktzugriffsdatei.

Der OPEN-Befehl muß vor allen Ein- und Ausgabebefehlen einer Datei stehen. Es ist jederzeit möglich mehrere Ein- und Ausgabedateien auf einmal zu öffnen.

Der **PRINT#**-Befehl
Mit dem PRINT#-Befehl werden die Daten in die Datei geschrieben. Dabei entspricht das Format der Daten genau dem Format des PRINT-Befehls, der Daten auf den Bildschirm schreibt. Eine Liste von Ausdrücken muß also mit Semikolon getrennt werden oder mit Kommata falls man Leerstellen zwischen den Daten wünscht.

Der **PRINT# USING**-Befehl
Er entspricht weitgehend dem PRINT#-Befehl, mit dem einzigen Unterschied, daß die Daten formatiert ausgegeben werden können.

Der **WRITE#**-Befehl
Dieser Befehl schreibt ebenfalls Daten in eine sequentielle Datei. Der Unterschied zu PRINT# besteht darin, daß WRITE# Kommata zwischen die Angaben einfügt, wie sie geschrieben werden und Zeichenketten (Strings) in Anführungszeichen einschließt. Aus diesem Grund braucht der Anwender keine Trennzeichen in diese Liste einzufügen. Außerdem werden auch keine Leerstellen vor eine positive Zahl gesetzt. Ein Wagenrücklauf bzw. Zeilenvorschub wird nach der letzten Angabe der Liste geschrieben.

Aufbau und Zugriff auf eine sequentielle Datei

Nachdem nun ausführlich die Befehle besprochen wurden, die für eine sequentielle Datei notwendig sind, soll jetzt gezeigt werden, wie man eine solche Datei aufbaut. Dazu sind folgende Schritte nötig:

1. Die Datei muß mittels des OPEN-Befehls in dem Modus "O" für eine Ausgabe geöffnet werden.
2. Die Daten müssen mittels der Anweisungen PRINT#, WRITE# oder PRINT# USING in die Datei geschrieben werden.
3. Soll nun auf die Daten zugegriffen werden, muß man zuerst die Datei mit dem CLOSE-Befehl schließen und anschließend mit dem OPEN-Befehl in dem Modus „I" für Eingabe geöffnet werden.
4. Jetzt kann man mit den Anweisungen INPUT# oder LINE INPUT# Daten aus der sequentiellen Datei lesen.

In Listing 1 wird dazu ein kurzes Beispiel gegeben.

Direktzugriffsdateien

Direktzugriffsdateien sind Dateien mit wahlfreiem Zugriff, d. h. man kann auf jeden einzelnen Datensatz auf der Diskette zugreifen und diesen anschließend bearbeiten. Um dies zu ermöglichen, ist jedem dieser Datensätze eine sogenannte Datensatznummer zugeordnet. Für Erstellung von Direktzu-

(Fortsetzung Seite 46)

```
List of \SEQDATEI.BAS

  10   '------Sequentielle Datei ------
  20   '
  30   closew 3:clearw 2:fullw 2
  40   dim ns$(20),nl$(20)
  50   schreiben:
  60   x=0
  70   open "O",#1,"Daten.seq"
  80   gotoxy 1,1:print
  90   input "Name : ";ns$(x)
 100   if ns$(x)="*" then goto 140
 110   x=x+1
 120   print
 130   goto 90
 140   print
 150   print "Bitte Datendiskette einlegen und Taste druecken !"
 160   a=inp(2)
 170   for i=0 to x-1
 180   print#1,ns$(i)
 190   next i
 200   close 1:print
 210   '
 220   '---------------------------------------------------------
 230   '
 240   lesen:
 250   x=0
 260   open "I",#1,"Daten.seq"
 270   if eof(1) then close 1:end
 280   input#1,nl$(x)
 290   print
 300   print "gelesener Name : ";nl$(x)
 310   x=x+1
 320   goto 270
```

Listing 1: Beispiel für eine sequentielle Datei

GEM

Was ist GEM?

Das Betriebssystem das ATARI ST nennt sich TOS. Es ist nach dem Chef von ATARI benannt (Tramiel Operating System). Das TOS ist eine Neuentwicklung von Digital Research, obwohl es in einigen Punkten Ähnlichkeit mit bereits bekannten Betriebssystemen wie CP/M-68K, MS-DOS oder auch UNIX besitzt. Doch was ist eigentlich ein Betriebssystem? Ein Betriebssystem ist ein Programm, also Software, die beim ATARI ST zur Zeit noch von Diskette geladen werden muß, abgesehen von einem kleinen Urlader (Bootprogramm), welches in sogenannten ROMs im Rechner „steckt". Erst durch diesen Urlader und durch das Betriebssystem (auf Diskette) wird der Rechner zum „Leben" erweckt, ist er überhaupt in der Lage andere Programme bzw. Befehle auszuführen. Das TOS unterteilt sich, wie die meisten modernen Betriebssysteme, in zwei große Teilbereiche, den eigentlichen Kern und eine Schale (auch Shell genannt), die den Kern umgibt. Der Kern des Betriebssystems, das eigentliche BIOS (Basic Input Output System), verwaltet die Hardware des Rechners einschließlich der Peripherie wie Diskette, Drucker, Tastatur etc.. Der „normale" Anwender hat mit diesem Kern nichts zu tun, kann ihn also kaum erreichen. Maschinennahe Programmierung, wie z. B. Assembler, bildet hier natürlich eine Ausnahme. Der zweite Teil des Betriebssystems die Schale oder auch Benutzeroberfläche liegt über dem Kern. Sie erledigt die Kommunikation mit dem Benutzer, empfängt seine Eingaben mittels der Tastatur oder der Maus. Diese Eingaben werden von der „Schale" interpretiert und ein entsprechender Befehl an den Kern des Betriebssystems weitergeleitet. Danach wartet sie auf ein Ergebnis vom Kern und meldet es dem Benutzer.

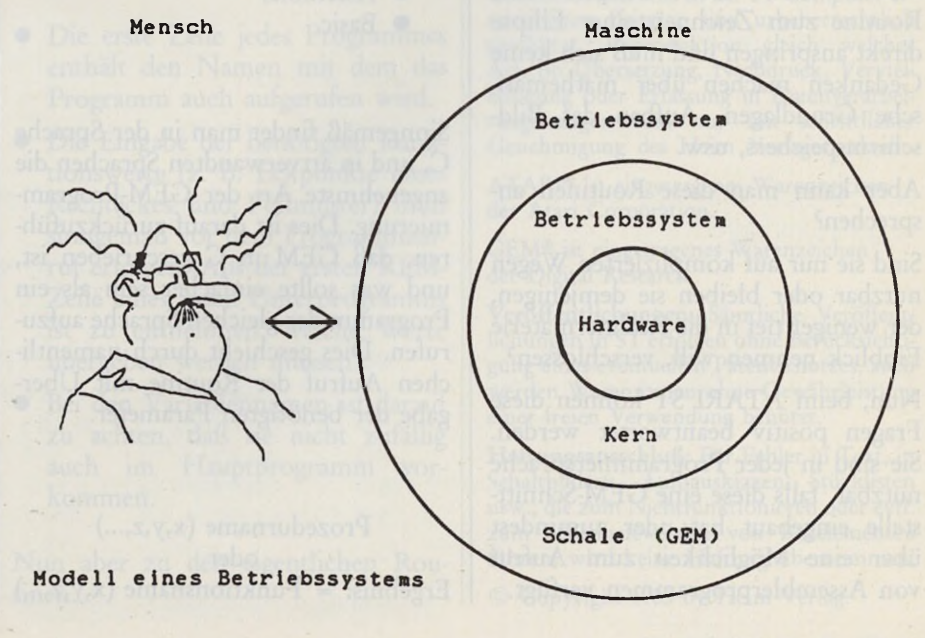
Beim ATARI ST wird diese Schale bzw. Benutzeroberfläche GEM oder auch GEMDOS genannt. GEM ist die Abkürzung für „Graphics Environment Manager" was etwa soviel bedeutet wie „Verwalter der grafischen Benutzeroberfläche". GEM verwaltet den hardwareunabhängigen Teil des Betriebssystems, es ist also ein Bindeglied (Schnittstelle) zwischen Mensch und Maschine. GEM setzt jedoch in seiner Form völlig neue Maßstäbe, die den heutigen benutzerfreundlichen Erfordernissen gerecht wird.

Entstehung von GEM

Der Grundstein für diese Art von „grafischer Benutzerführung" wurde bereits 1970 von der Firma Xerox gelegt. Während Xerox ein völlig neues, hochwertiges und kompliziertes System für den Bereich der Bürokommunikation zu entwickeln versuchte, gelang es ein wenig später dem amerikanischen Computerhersteller APPLE einen „preiswerten" Computer für jedermann, namens Lisa, zu entwickeln. Lisa erschien mit einer grafischen Benutzeroberfläche, nach Art von Xerox, Anfang der 80er Jahre. Mangels geeigneter Software, die diese neue Benutzeroberfläche unterstützte, kündigte sich jedoch bald der kommerzielle Tod von Lisa an. Erst durch den Macintosh, APPLES neusten Computer, wurde Lisa's Konzept der grafischen Benutzerführung bekannt. Merkmale dieser grafischen Benutzeroberfläche, die mit einer Maus bedient wird, sind die Verwendung von Fenstern (Windows), sogenannten Icons (grafische Umsetzung von Objekten wie z. B. einem Papierkorb oder Aktenordner usw.) und Pull-Down-Menüs (kleine Fenster, die auf Wunsch des Benutzers Informationen in das laufende Programm einblenden). GEM, wie es mit dem ATARI ST geliefert wird, stellt zur Zeit den letzten Schritt in dieser Entwicklung dar. Es ist größtenteils in der Programmiersprache „C" geschrieben.

Bestandteile von GEM

GEM beinhaltet aber nicht nur die einfach zu bedienende und verbesserte Betriebssystemoberfläche des Macintosh's, sondern auch ein riesiges Bibliotheks-Paket mit vielen nützlichen Routinen, die der Programmierer auch für seine eigenen Programme nutzen kann und bei Bedarf auch sollte. So sind z. B. fertige Routinen für



Modell eines Betriebssystems

die Eingabe über Tastatur oder Maus, die Ausgabe von Zeichen bzw. Texten auf Bildschirm oder Drucker ebenso vorhanden, wie zum Betrieb der Schnittstellen und der Diskettenbehandlung. Einige dieser Routinen werden in der sogenannten GEM-AES Schicht aufgerufen. AES ist die Kurzform für Application Environment Services, was frei übersetzt soviel bedeutet wie „Hilfsmittel zur Gestaltung von Anwenderprogrammen". Wie eine Routine dann mit der Hardware des Rechners zusammenarbeitet braucht den Anwender nicht zu kümmern, denn dies besorgt die GEM-VDI Ebene. VDI ist die Abkürzung für Virtual Device Interface; zu deutsch etwa „imaginäre Geräteschnittstelle". Interessant ist die Möglichkeit solche Routinen von anderen höheren Programmiersprachen aus aufzurufen. Wie man z. B. von Basic aus verschiedene Zeichensätze aufrufen oder die Strichstärke einer Linie verändern kann, ist ein wenig später erklärt.

Fühlen Sie sich jedoch durch die Bedienung mit der Maus oder der Fenstertechnik des GEMs behindert, so haben Sie beim ATARI ST die Möglichkeit auf das TOS ohne die GEM-Ebene umzuschalten (siehe Anwendung anmelden). Dann erfolgen alle Eingaben mittels herkömmlicher Kommandos, ähnlich MS-DOS, über die Tastatur.

GEM-Routinen

Prinzipielle Nutzung

Basicerweiterung in Form von Unterprogrammen

GEM enthält eine Vielzahl von leistungsstarken Routinen, die vor allem bei grafikorientierten Programmen ihre Anwendung finden. Der Programmierer kann zum Beispiel eine Routine zum Zeichnen einer Ellipse direkt anspringen und muß sich keine Gedanken machen über mathematische Grundlagen, Aufbau des Bildschirmspeichers, usw.

Aber kann man diese Routinen ansprechen?

Sind sie nur auf komplizierten Wegen nutzbar oder bleiben sie demjenigen, der weniger tief in die Rechnermaterie Einblick nehmen will, verschlossen?

Nun, beim ATARI ST können diese Fragen positiv beantwortet werden. Sie sind in jeder Programmiersprache nutzbar, falls diese eine GEM-Schnittstelle eingebaut hat oder zumindest über eine Möglichkeit zum Aufruf von Assemblerprogrammen verfügt.

Dazu gehören momentan:

- C
- Pascal, Modula 2, BCPL, ...
- Assembler
- Basic

Singgemäß findet man in der Sprache C, und in artverwandten Sprachen die angenehmste Art der GEM-Programmierung. Dies ist darauf zurückzuführen, daß GEM in C geschrieben ist, und was sollte einfacher sein als ein Programm der gleichen Sprache aufzurufen. Dies geschieht durch namentlichen Aufruf der Routine mit Übergabe der benötigten Parameter.

Prozedurname (x,y,z,...)
oder
Ergebnis: = Funktionsname (x,y,z,...)

Da jede Routine über einen eigenen Namen verfügt, ist der Funktionsaufruf besonders übersichtlich, wobei in C auf diese Art und Weise alle GEM-Routinen in ein eigenes Programm eingebaut werden können. Andere höhere Programmiersprachen bieten ähnliche Möglichkeiten. Natürlich ist GEM auch in Assembler ansprechbar. Dies erreicht man durch Angabe einer fest zugeordneten Funktionsnummer und einen anschließenden speziellen Assemblerbefehl, wobei die Lage der angesprochenen Routine im Speicher, wie bei manch anderem Betriebssystem, keine Rolle spielt.

Auch in Basic ist die Einbeziehung von GEM vorgesehen, zwar nicht so komfortabel wie in C, aber immerhin möglich. Die Handhabung ist ähnlich wie in Assembler gestaltet. Es stehen hier vordefinierte Speicherzellen zur Verfügung, in denen die benötigten Befehlsparameter übergeben werden.

Bedeutung der Speicherzellen bzw. Datenfelder

Eingabe:

CONTRL	Befehlsnummer
CONTRL+2	Anzahl der Eingabekoordinaten des PTSIN-Feldes
CONTRL+6	Anzahl der Eingabedaten im INTIN-Feld
INTIN	Eingabefeld (Farbe, Schriftart, ...)
PTSIN	Eingabefeld von Datenpaaren (z. B. Eckpunkte von Rechtecken)

Ausgabe:

CONTRL+4	Anzahl der Ausgabekoordinaten
CONTRL+8	Anzahl der Ausgabedaten
INTOUT	Ausgabefeld (z. B. momentaner Schrifttyp)
PTSOUT	Ausgabefeld von Datenpaaren (Eckpunkte eines Buchstabens)

Sämtliche Speicherzellen sind 16-Bit breit, was bei der Adressberechnung berücksichtigt werden muß. Das bedeutet, daß eine Datenzelle 'CONTRL+1' nicht existiert. Falls die Bitbreite der Befehle PEEK und POKE abgeändert wurde, sollte durch 'DEF SEG = 0' während der Bearbeitung der GEM-Routine die Bitbreite wieder auf 16-Bit eingestellt werden.

Tabelle 1 : Bedeutung der Variablen

Durch die Basicbefehle GEMSYS und VDISYS erfolgt der Funktionsaufruf von AES- und VDI-Routinen. In diesem Artikel soll näher auf die VDI-Routinen eingegangen werden, da diese von Basic aus wesentlich einfacher zu handhaben sind.

Einige dieser Routinen sind schon als Basicbefehle eingebaut. Es sind die grafischen Grundfunktionen wie zum Beispiel das Ziehen von Linien, das Zeichnen von Ellipsen oder Teilkreisen sowie das Ausfüllen von Flächen. Allerdings sind einige leistungsstarke grafische Möglichkeiten des Rechners dabei verloren gegangen. Diese doch recht einfach realisierbare Funktionen kann man aber leicht durch Unterprogramme ersetzen.

Prinzip des GEM-VDI Aufrufs

Der Aufruf von VDI erfolgt immer nach dem gleichen Prinzip.

```
WERT1 = num. Wert:
WERT2 = num. Wert:
...
GOSUB NAME
```

Gestaltung der Unterprogramme

Sämtliche Basicunterprogramme sind so gestaltet, daß man sie nach Bedarf an jedes beliebige Programm anhängen (MERGEN) kann. Es ist daher empfehlenswert die Zeilennummern zu übernehmen.

- Die erste Zeile jedes Programmes enthält den Namen mit dem das Programm auch aufgerufen wird.
- Die Eingabe der benötigten Funktionswerte (z. B. Eckpunkte eines Rechteckes und Füllmuster) muß singgemäß vor dem Programmaufruf erfolgen. Aus der ersten REM-Zeile eines jeden Unterprogramms ist zu entnehmen, welche Werte übergeben werden müssen.
- Bei den Variablennamen ist darauf zu achten, daß sie nicht zufällig auch im Hauptprogramm vorkommen.

Nun aber zu den eigentlichen Routinen.

Impressum

ST-Computer

Herausgeber: Heim Fachverlag, Heidelberger Landstraße 194, 6100 Darmstadt 13, Telefon (0 61 51) 5 53 75

Redaktion: Uwe Bärtels (UB-Chefredakteur), Harald Schneider (HS), Marcelo Merino (MM), Harald Egel (HE)

Uwe Bärtels
ST Redaktion
Postfach 11 31
6242 Kronberg

Freie Mitarbeiter: Markus Nerding (MN), Jörg Mainusch (JM), Wilfried Rüsse (WR), Volker Sehorz (VS)

Titelseite: Klaus Ohlenschläger

Anzeigen: Anzeigenleiter H. Heim, Telefon (0 61 51) 5 53 75
Anzeigenpreise nach Preisliste Nr. 1 gültig ab 1.1.86

Erscheinungsweise: 11 x jährlich

Bezugspreis: Einzelheft DM 6,–. Jahresabonnement DM 60,– inklusive der gesetzlichen Mehrwertsteuer und den Zustellgebühren für 11 Ausgaben.

Bezugsmöglichkeiten: ATARI-Fachhändler, Zeitschriftenhandel, Kauf- und Warenhäuser oder direkt beim Verlag unter obiger Adresse.

Druck: Ferling Druck Darmstadt

Manuskripteinsendungen: Programmlistings, Bauanleitungen und Manuskripte werden von der Redaktion gerne angenommen. Sie müssen frei von Rechten Dritter sein. Mit ihrer Einsendung gibt der Verfasser die Zustimmung zum Abdruck und der Vervielfältigung auf Datenträgern im Heim Verlag. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte wird keine Haftung übernommen.

Schriftart

Der ATARI ST verfügt über verschiedene Textdarstellungsformen. Dazu gehören Fettschrift, Kursivschrift, Hohlschrift, unterstrichene und helle Schrift oder beliebige Kombinationen. Dabei schalten Bit 0 bis Bit 4 der 'SCHRIFTTYP'-Variablen die verschiedenen Schriftarten.

BIT	Wert	Schriftart
0	1	Fettschrift
1	2	Helle Schrift
2	4	Kursivschrift
3	8	Unterstrichen
4	16	Hohlschrift

Durch Kombination kommt man schon auf 32 verschiedene Schriftarten bei denen allerdings einige schlecht lesbar sind.

Der Aufruf erfolgt durch:

SCHRIFTTYP = ...
Gosub SCHRIFTART

(0 < = SCHRIFTTYP < = 32)

Eine Auswahl der verschiedenen Schriftarten zeigt Bild 1 (dazugehörend Listing 1).

Der normale Schriftsatz wird wieder eingeschaltet durch:

SCHRIFTTYP = 0 :
Gosub SCHRIFTART

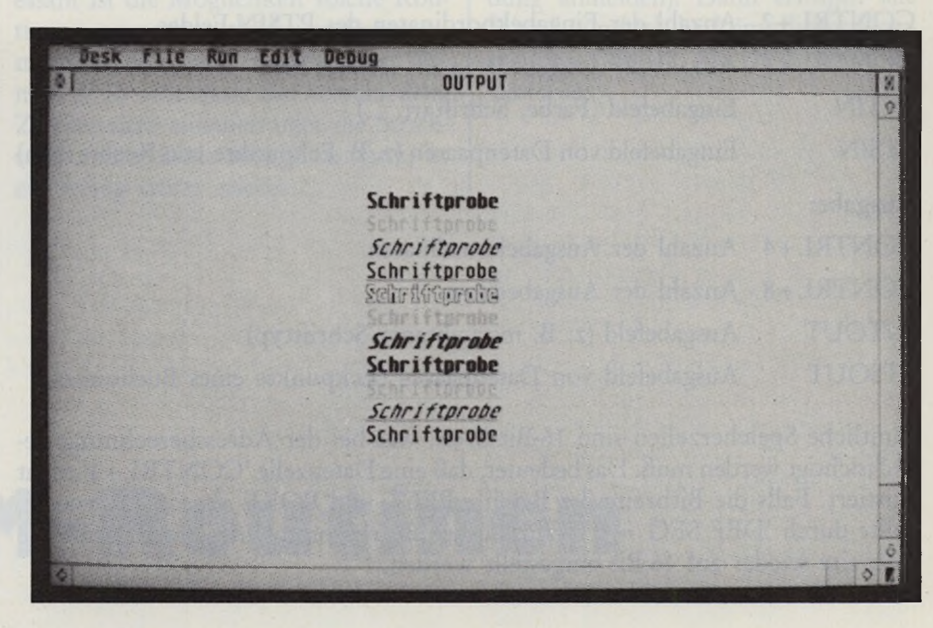


Bild 1

```
65000 SCHRIFTART:
65002 ' --> SCHRIFTTYP
65004 '     0 = Normal       1 = Fett       2 = Hell
65006 '     4 = Kursiv       8 = Unterst.  16 = Hohl
65008 '                   oder Kombinationen
65010 '
65012 poke contrl ,106
65014 poke contrl+2,0
65016 poke contrl+6,
65018 poke intin ,schrifttyp
65020 vdisys
65022 return
```

Routine Schriftart

```
5       ' merge "SCHRIFTART.bas"
9       '
10      fullw 2: clearw 2
20      data  1,2,4,8,16,3,5,9,10,12,0
30      for a=0 to 10
40      read SCHRIFTTYP
50      gosub SCHRIFTART
60      gotoxy 14,a+4:    print "Schriftprobe"
70      next
80      warte=inp(2)
90      end
```

Listing 1: verschiedene Schriftarten

Schriftgröße

Ebenso kann man die Größe der Schrift verändern (Bild 2). Die Schriftgrößen reichen hierbei von Kleinstschrift, wie man sie in den Diskettensymbolen findet, bis zu äußerst großer Schrift, deren Ausmaße ein Vielfaches der Normalschrift einnehmen.

Der Aufruf erfolgt durch:

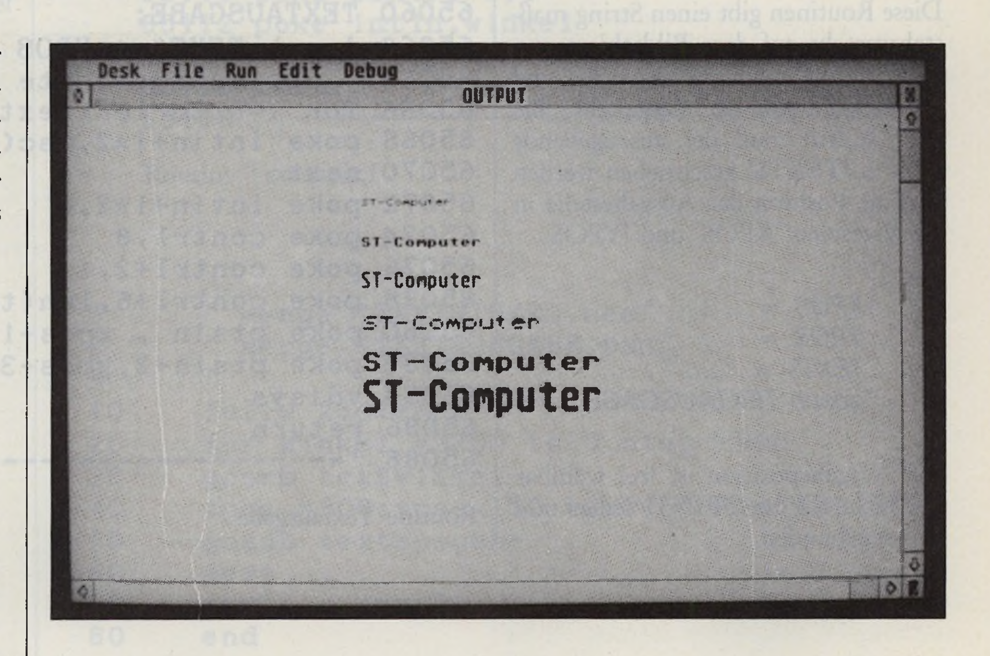
GROESSE = ... :
gosub SCHRIFTGROESSE

Dabei ist zu beachten, daß einige Schriftgrößen nicht mehr mit dem PRINT-Befehl ausgegeben werden können, da sich sonst die einzelnen Buchstaben bei großer Schrift überschneiden und bei kleiner Schrift ein zu großer Zwischenraum zwischen den Buchstaben entsteht. Vermeiden kann man dies durch Ausgabe des Textes mit einer weiteren VDI-Routine (Siehe Routine 'Textausgabe').

Bei dieser Routine ist darauf zu achten, daß die Schriftgröße nach Programmablauf wieder auf Normalgröße zurückgeschaltet wird, da es sonst zu Schwierigkeiten bei der Programmeditierung kommt.

Dies geschieht durch:

GROESSE = 10 :
gosub SCHRIFTGROESSE




```
65030 SCHRIFTGROESSE:
65032 ' --> GROESSE
65034 '    <9  sehr klein      9   klein
65036 ' 10-15 normal        16-17 gross
65038 ' 16-17 gross         18-19 groesser
65040 '
65042 poke contrl ,107
65044 poke contrl+2,0
65046 poke contrl+6,1
65048 poke intin ,groesse
65050 vdisys
65052 return
```

Routine Schriftgröße

```
5     ' merge "SCHRIFTGROESSE.bas"
6     ' merge "TEXTAUSGABE.bas"
9     '
10    fullw 2:clearw 2
20    data 8,9,10,16,18,20
30    for a=1 to 6
40    read groesse
50    gosub schriftgroesse
60    xpos=220:ypos=80+a*30:text$="ST-Computer"
70    gosub textausgabe
80    next
90    groesse=10 : gosub schriftgroesse
100   warte=inp(2)
110   end
```

Listing 2: verfügbare Schriftgrößen

Textausgabe

Diese Routinen gibt einen String maßstabsgerecht auf dem Bildschirm aus. Dabei wird auch Großschrift und Kleinschrift korrekt dargestellt. Bei dem Aufruf muß der auszugebende Text in 'TEXT$' geschrieben werden und die Position der Ausgabestelle in die Variablen 'XPOS' und 'YPOS'.

XPOS = ... :
YPOS = ... :
TEXT$ = "....." :
gosub TEXTAUSGABE

Die Ausgabeposition ist frei wählbar und nicht wie bei 'PRINT' zeilen- und spaltengebunden.

```
65060 TEXTAUSGABE:
65062 ' --> TEXT$ ; XPOS ; YPOS
65064 ' masstabsgerechte Textausgabe
65066 for i=0 to len(text$)-1
65068 poke intin+i*2,asc(mid$(text$,i+1,1))
65070 next
65072 poke intin+i*2,0
65074 poke contrl,8
65076 poke contrl+2,1
65078 poke contrl+6,len(text$)+1
65080 poke ptsin , xpos+1
65082 poke ptsin+2,ypos+38
65084 vdisys
65086 return
65088 '-----------------------------------
```

Routine Textausgabe

Textwinkel

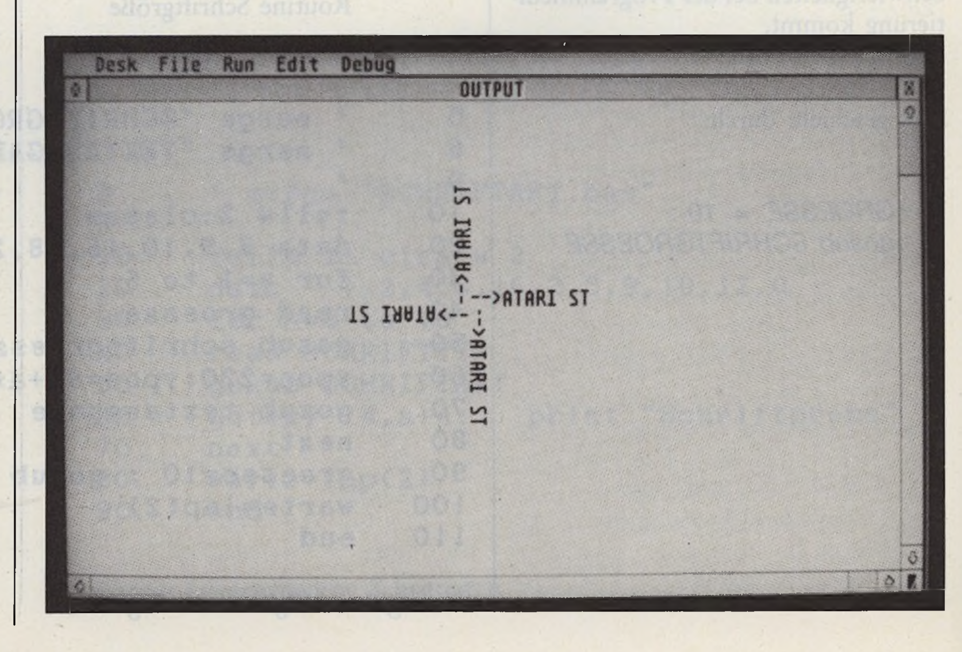
Textwinkel

Als weiteren Leckerbissen kann man die Ausgaberichtung der Schrift verändern (Bild 3 und Listing 3). Leider erfolgt dies nicht kontinuierlich sondern nur in 90 Grad Schritten. Anwendung findet beispielsweise die dadurch erreichbare senkrechte Textausgabe bei der Beschriftung von Koordinatensystemen oder bei der Bemaßung von Zeichnungen.

Aufruf durch:

WINKEL = ... :
gosub TEXTWINKEL

(Winkel = 0 ; 900 ; 1800 ; 2700)



Die Routine arbeitet nur mit der GEM-Textausgabe zusammen, da der normale PRINT-Befehl dafür nicht ausgelegt ist. Wie schon bei den vorhergehenden Routinen ist auch hier darauf zu achten, daß man vor Beendigung des Programmes den Winkel wieder normalisiert.

Dies erfolgt durch:

WINKEL = 0 :
gosub TEXTWINKEL

```
65100 TEXTWINKEL:
65102 ' --> WINKEL
65104 '      (0 ; 900 ; 1800 ; 2700 )
65106 poke contrl ,13
65108 poke contrl+2,0
65110 poke contrl+6,1
65112 poke intin,winkel
65114 vdisys
65116 return
65118 ' ----------------------------
```

Routine Textwinkel

```
5     ' merge "TEXTWINKEL.bas"
6     ' merge "TEXTAUSGABE.bas"
9     '
10    fullw 2:clearw 2
20    for winkel =2700 to 0 step -900
30    gosub textwinkel
40    xpos =300:ypos=150:text$="-->ATARI ST"
50    gosub textausgabe
60    next
70    warte = inp(2)
80    end
```

Listing 3.

Grafikmodus

Diese Funktion gibt den Schreibmodus an, der bei allen grafischen Operationen von Bedeutung ist. Hiermit wird festgelegt, ob der Bildschirm überschrieben wird oder nach einer anderen bestimmten logischen Anweisung verknüpft wird. Es entstehen auf diese Art und Weise vier verschiedene Möglichkeiten der Bildschirmausgabe:

Modus	Bedeutung
1	Überschrieben
2	Mischen
3	XOR
4	Mischen, Revers

Der Aufruf erfolgt durch:

MODUS = 1...4 :
Gosub GRAFIKMODUS

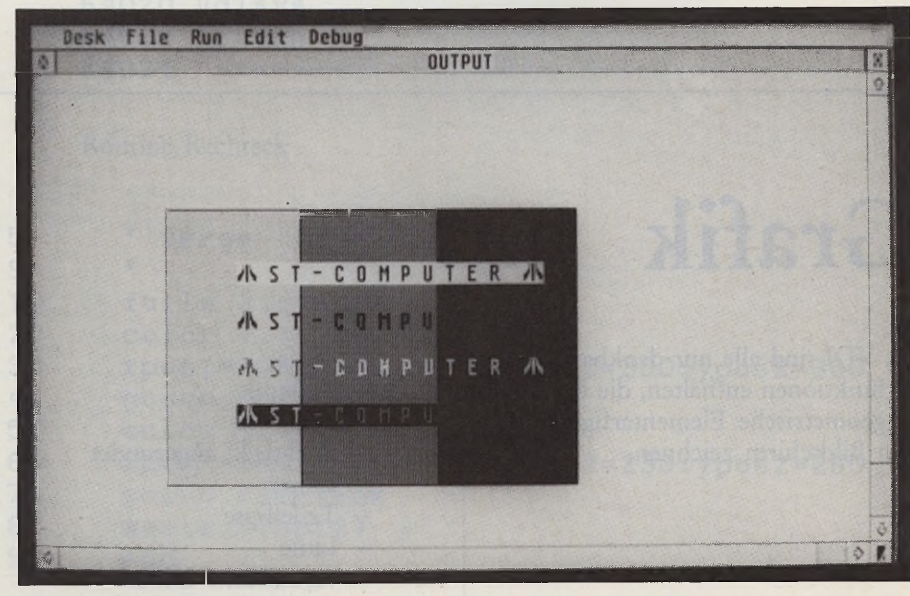


Bild 4

Modus 1 ist der Schreibmodus, der normalerweise bei allen Bildschirmausgaben in Erscheinung tritt. Der Hintergrund wird hierbei grundsätzlich gelöscht.

In Modus 2 werden die Bildschirmausgaben mit dem schon vorhandenen Hintergrund gemischt. Dies ist vor allem bei Beschriftungen von Grafiken interessant, da im normalen Schreibmodus hinter jedem Schriftzeichen eine leere Box entsteht.

Eine exclusive Oder-Verknüpfung erreicht man durch Modus 3. Ein Bildschirmpunkt wird hierbei nur dann gesetzt, wenn eines der beiden Bits (Hintergrundbit oder Objektbit) eine Eins enthält. Der betreffende Bildschirmpunkt wird gelöscht, wenn beide Bits Nullen oder beide Bits Einsen enthalten.

Der letzte mögliche Modus ist Modus 4. Bei ihm erfolgt die Bildschirmausgabe wie bei Modus 2, jedoch mit zusätzlicher Invertierung. Damit kann z. B. Reversschrift ausgegeben werden:

MODUS = 4 :
Gosub GRAFIKMODUS

Bild 4 zeigt die verschiedenen Wirkungen des 'Grafikmodus'-Befehls auf unterschiedlichen Hintergründen.

```
5     ' merge "GRAFIKMODUS.bas"
9     '
10    fullw 2:clearw 2
20    linef 100,100,400,100
30    linef 100,100,100,300
40    linef 100,300,400,300
50    linef 200,100,200,300
60    linef 400,100,400,300
70    linef 300,100,300,300
80    color 1,1,1,2,2 : fill 110,110
90    color 1,1,1,6,2 : fill 210,110
100   color 1,1,1,8,2 : fill 310,110
110   for MODUS =4 to 1 step-1
120   gosub GRAFIKMODUS
130   gotoxy  9, 6 + modus*2
140   print chr$(14) chr$(15);
150   print " S T - C O M P U T E R  ";
160   print chr$(14) chr$(15)
170   next
180   warte=inp(2)
190   end
```

Listing 4: Grafikmodi

```
65130 GRAFIKMODUS:
65132 ' --> MODUS
65134 '     1 = ueberschreiben        2 = mischen
65136 '     3 = XOR-verknuepfen       4 = revers,mischen
65138 poke contrl ,32
65140 poke contrl+2,0
65142 poke contrl+6,1
65144 poke intin ,modus
65146 vdisys
65148 return
65150 '-----------------------------------------
```

Routine Grafikmodus

Grafik

Im VDI sind alle nur denkbaren Grafikfunktionen enthalten, die bestimmte geometrische Elementarfiguren auf den Bildschirm zeichnen.

- Kreis
- Teilkreis
- Rechteck
- Rechteck, abgerundet
- Ellipse
- Teilellipse
- Linie
- verbundene Linien
- ausgefüllte Flächen

Einige dieser Funktionen sind direkt im Basic verfügbar, andere sind allerdings nur über GEM Aufrufe erreichbar. Die interessanten, bzw. die am meisten benötigten Routinen sind hier angegeben.

Rechteck

Leider vermißt man im Basic einen Befehl zum Zeichnen von Rechtecken. Diesen kann man zwar mit vier LINE-Befehlen umgehen, aber einfacher und schneller ist die spezielle VDI-Routine des GEM hierfür. Damit kann man das Rechteck zusätzlich mit einem vorgewählten Füllmuster ausfüllen lassen, was wesentlich schneller ausgeführt wird, als mit dem FILL-Befehl (Bild 5 und Listing 5).

Zum Aufruf genügt die Angabe der Koordinaten von zwei gegenüberliegenden Eckpunkten. Die Reihenfolge der Eckkoordinaten ist dabei völlig beliebig. Es ist also nicht erforderlich immer die linke obere Ecke und anschließend die rechte untere Ecke anzugeben. Die Funktion verlangt lediglich zwei Punkte die an einer Diagonalen des Rechteckes liegen.

Der Aufruf erfolgt durch:

XPOS1 = ... : YPOS1 = ... :
XPOS2 = ... : YPOS2 = ...
Gosub RECHTECK

Bei dieser Routine, wie auch bei allen anderen GEM-Routinen, die etwas auf den Bildschirm ausgeben, sollte man darauf achten, daß das OUTPUT-Fenster in voller Größe geöffnet ist, denn anders wie bei dem Print-Befehl erfolgt die Ausgabe bei VDI-Befehlen direkt auf den Bildschirm, d. h. in die Fenster, die sich gerade auf dem Bildschirm befinden.

Die Koordinatenumrechnung ist erforderlich weil sich der normale Koordinatenursprung des Basic-Koordinatensystems wegen des Output-Fensters unterscheiden. Die Abweichung beträgt in X-Richtung 1 Pixel und in Y-Richtung 38 Pixels. Der Angleich wurde vorgenommen damit man problemlos VDI- und normale Basicroutinen kombinieren kann.

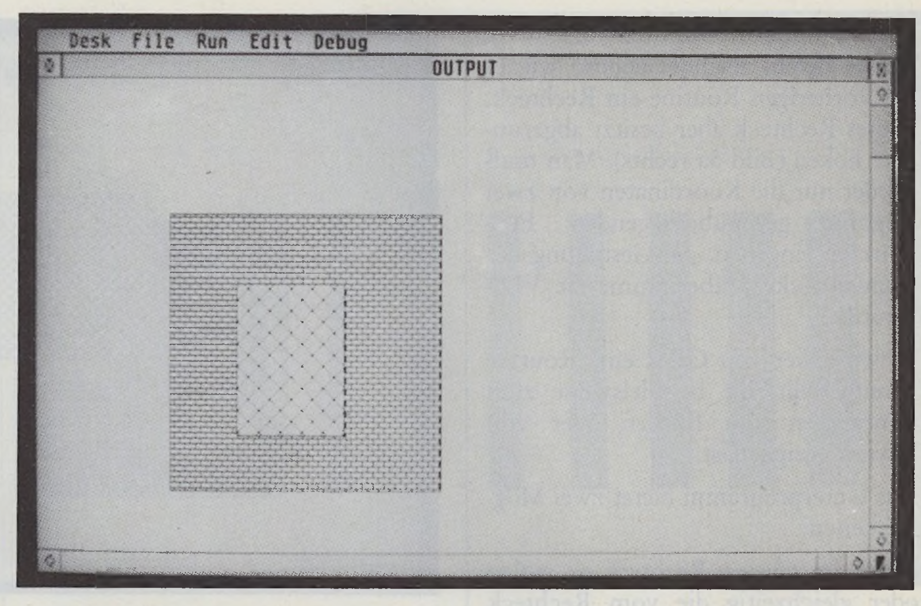


Bild 5

```
64000 RECHTECK:
64002 '--> XPOS1 ; YPOS1 ; XPOS2 ; YPOS2
64004 poke contrl    ,11
64006 poke contrl+2  ,2
64008 poke contrl+6  ,0
64010 poke contrl+10,1
64012 poke ptsin     ,XPOS1 +1
64014 poke ptsin+2,YPOS1 +38
64016 poke ptsin+4,XPOS2 +1
64018 poke ptsin+6,YPOS2 +38
64020 vdisys
64022 return
64024 '----------------------------------
```

Routine Rechteck

```
5     ' merge "RECKTECK.bas"
9     '
10    fullw 2:clearw 2
20    color 1,1,1,9,2
30    xpos1=100:ypos1=100:xpos2=300:ypos2=300
40    gosub rechteck
50    color 1,1,1,9,3
60    xpos1=150:ypos1=150:xpos2=230:ypos2=260
70    gosub rechteck
80    warte =inp(2)
90    end
```

Listing 5: Rechtecke, eckig

Rechteckrund

Rechteck abgerundet

Dieser Befehl zeichnet ebenso wie in der vorherigen Routine ein Rechteck. Dieses Rechteck aber besitzt abgerundete Ecken (Bild 5a rechts). Man muß wieder nur die Koordinaten von zwei diagonal gegenüberliegenden Eckpunkten eingeben, die Gestaltung der 'runden Ecken' übernimmt die VDI-Routine.

Es wird hier von GEM eine Routine bereitgestellt, die beispielsweise zum Umrahmen von Bildern oder von Texten geeignet ist.

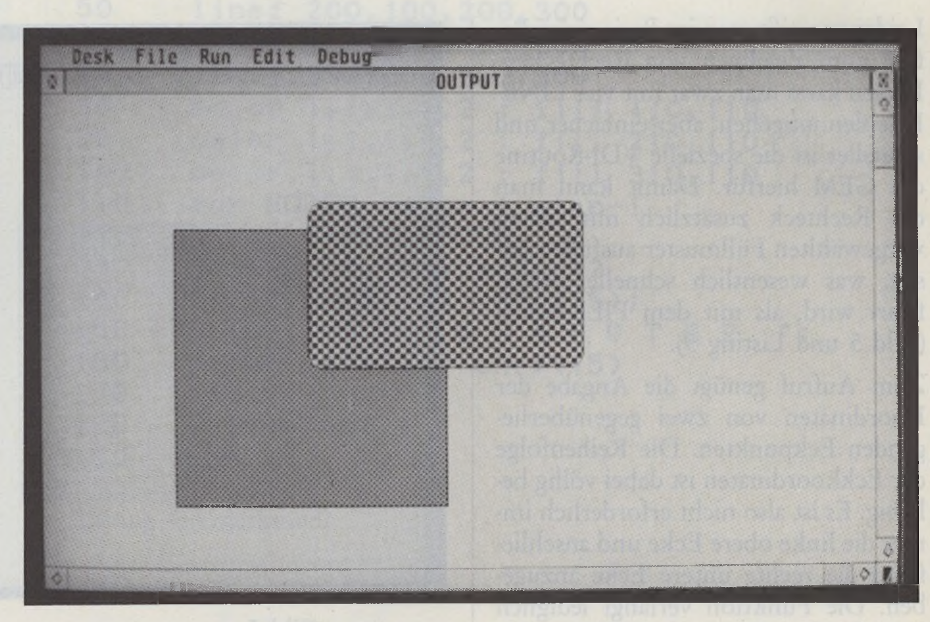
Das Unterprogramm bietet zwei Möglichkeiten:

Entweder nur ein Rechteck zu malen oder gleichzeitig die vom Rechteck eingeschlossene Fläche auszufüllen.

Der Aufruf erfolgt durch:

XPOS1 = ... : YPOS1 = ... :
XPOS2 = ... : YPOS2 = ... :
FUELL = ... :
gosub RECHTECKRUND

Die Variable FUELL gibt an ob ein Rechteckrahmen oder ein ausgefülltes Rechteck gezeichnet werden soll. Dabei wird bei 'FUELL=0' ein Rahmen gemalt und bei 'FUELL<>0' ein ausgefülltes Rechteck. Die Füllfarbe bzw. das Füllmuster muß vorher mit dem COLOR-Befehl definiert werden.




```
64030 RECHTECKRUND:
64032 ' --> XPOS1 ; YPOS1 ; XPOS2 ; YPOS2
64034 ' --> FUELL ' (0 ODER <> 0)
64036 poke contrl ,11
64038 poke contrl+2,2
64040 poke contrl+6,0
64042 if fuell = 0 then poke contrl+10,8
                   else poke contrl+10,9
64044 poke ptsin  ,XPOS1 +1
64046 poke ptsin+2,YPOS1 +38
64048 poke ptsin+4,XPOS2 +1
64050 poke ptsin+6,YPOS2 +38
64052 vdisys
64054 return
64056 '------------------------------------
```

Routine Rechteckrund

```
5     ' merge "RECHTECK.bas"
6     ' merge "RECHTECKRUND.bas"
9     '
10    color 1,1,1,5 ,2
15    fullw 2:clearw 2
20    xpos1=100:ypos1=100:xpos2=300:ypos2=300
30    gosub rechteck
40    xpos1=200:ypos1=80:xpos2=400:ypos2=200
50    FUELL=1
60    color 1,1,1,22,2
70    gosub rechteckrund
80    warte =inp(2)
90    end
```

Listing 5a: Rechtecke

Linienstärke

Linienbefehle

Eine weitere Besonderheit des GEM-VDI ist die Veränderbarkeit der Linien. Dazu gehören:

- **Linienstärke**
- **Linienmuster**
- **Aussehen der Linienendpunkte**

Alle Grafikoperationen benutzen sehr feine, durchgehende Linien. Eine Änderung dieser Attribute ist im ST-Basic nicht vorgesehen. GEM-VDI bietet da einige Möglichkeiten, die bei der einen oder anderen Anwendung sehr nützlich sein können.

Dabei gelten diese Routinen nicht nur für Geraden sondern auch für Ellipsen und Rechtecke.

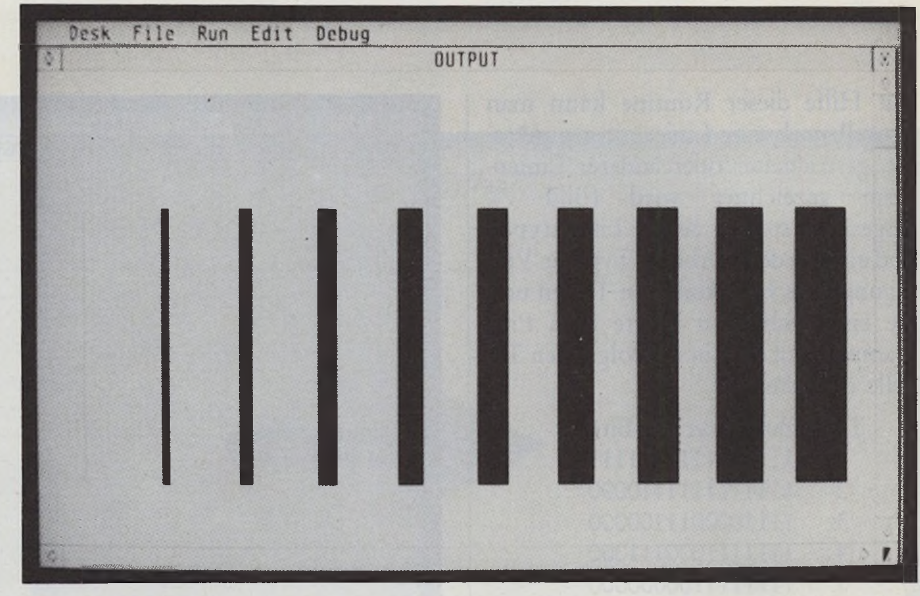


Bild 6

Linienstärke

Diese Routine gibt an mit welcher Stärke eine Linie auf dem Bildschirm abgebildet wird (Bild 6). Damit ist eine Hervorhebung bestimmter Kurven oder grafischen Figuren möglich.

Die Routine wird wie folgt aufgerufen:

STAERKE= ... :
Gosub LINIENSTAERKE

'STAERKE = (1,3,5,7,...)'

Zur Normalisierung der Linienstärke wird die Routine mit 'STAERKE = 1' aufgerufen.

```
64220 LINIENSTAERKE:
64222 ' --> STAERKE
64224 poke contrl ,16
64226 poke contrl+2,1
64228 poke contrl+6,0
64230 poke ptsin, staerke
64232 poke ptsin+2, 0
64234 vdisys
64236 return
64238 '--------------------
```

Routine Linienstärke

```
5    ' merge "LINIENSTAERKE.bas"
9    '
10   fullw 2: clearw 2
20   for staerke = 37 to 1 step -4
30   gosub linienstaerke
40   x=20+staerke*15
50   linef x,100,x,300
60   next
70   warte = inp(2)
80   end
```

Listing 6: verschiedenen Linienstärken

Linienmuster

Mit Hilfe dieser Routine kann man einstellen ob eine Linie mit gepunkteter, gestrichelter oder anderer Linienform gezeichnet wird (Bild 7). Insgesamt stehen sechs Linientypen und ein frei definierbarer Typ zur Verfügung. Die vordefinierten Typen und die entsprechenden Werte zum Programmaufruf sind in der folgenden Tabelle dargestellt.

Typ	Bitmuster (16-Bit)
1	1111111111111111
2	1111111111110000
3	1111000011100000
4	1111111000111000
5	1111111100000000
6	1111000110011000
7	frei definiertes Muster

Diese Muster eignen sich beispielsweise zum Darstellen von Hilfslinien in Diagrammen oder verdeckten Linien in dreidimensionalen Körpern. Auch zum gleichzeitigen Darstellen mehrerer Kurven sind verschiedene Linientypen, zumindest auf einem monochromen Monitor, unverzichtbar. Der Aufruf erfolgt durch:

LINIENTYP = ... :
Gosub LINIENMUSTER

Zum Wiederherstellen der normalen geschlossenen Linienform ruft man die Routine mit 'Linienmuster=1' auf.

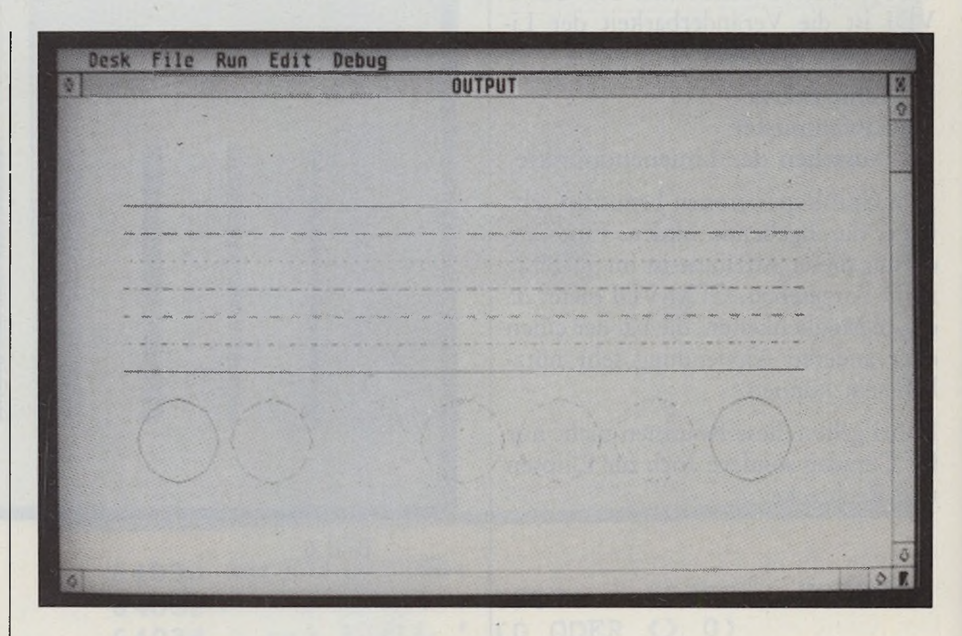


Bild 7

```
64240 LINIENMUSTER:
64242 ' --> LINIENTYP
64244 ' 1-7
64246 poke contrl ,15
64248 poke contrl+2,0
64250 poke contrl+6,1
64252 POKE INTIN, linientyp
64254 vdisys
64256 return
64258 '--------------------------------
```

Routine Linienmuster

```
5     ' merge "LINIENMUSTER.bas"
9     '
10    fullw 2:clearw 2
20    for linetyp = 1 to 7
30    gosub   linienmuster
40    y=60 +20*linetyp
50    linef 50,y,550,y
52    z=20 +70*linetyp
55    circle z,250,30
60    next
70    warte =inp(2)
80    end
```

Listing 7: Linienmuster

Linienendform

Normalerweise sehen alle Linien am Anfang und Ende eckig aus. Dies ist natürlich auch angebracht. Mit dieser Routine allerdings kann man dieses Aussehen verändern. Es stehen zwei zusätzliche Endformen zur Verfügung, wobei Anfangs- und Endform getrennt definiert werden.

- abgerundete Endform (2)
- Pfeil (1)
- eckig (0)

Die Zahl in Klammer gibt den Wert an, der in den Variablen übergeben werden muß.

Aufruf durch:

ANFANGSFORM = ... :
ENDFORM = ... :
Gosub LINIENENDFORM

Die Pfeilform ist hierbei sehr gut zur Bemaßung einzusetzen oder zum Darstellen von Vektoren. Die Wirkung dieser Routine zeigt Bild (8). Kombiniert wurde dieses Programm mit den Routinen 'LINIENSTAERKE' und 'GRAFIKMODUS'.

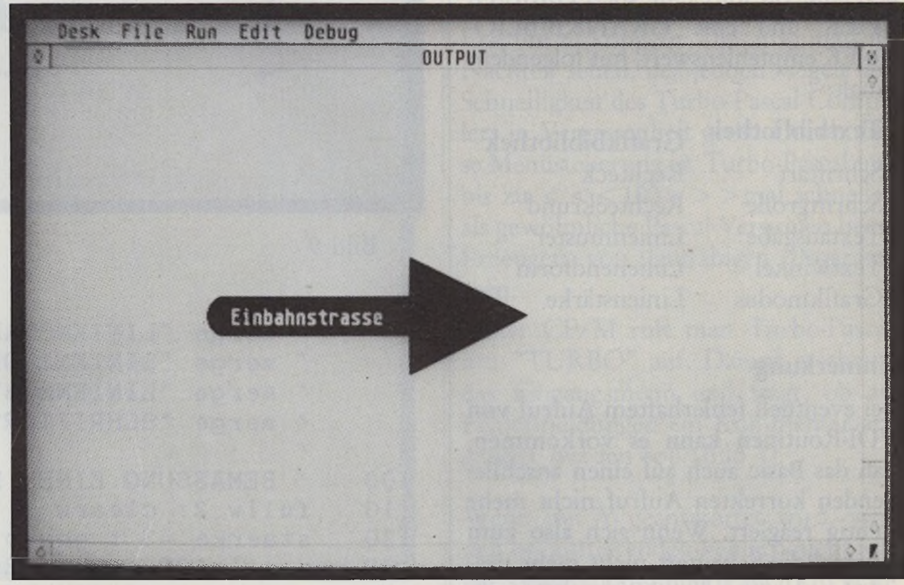


Bild 8

```
64200 LINIENENDFORM:
64201 '--> ANFANGSFORM ; ENDFORM
64202 poke contrl ,108
64204 poke contrl+2,1
64206 poke contrl+6,0
64208 poke intin, Anfangsform
64210 poke intin+2, Endform
64212 vdisys
64214 return
64216 '--------------------------
```

Routine Linienendform

```
5     'merge "LINIENENDFORM.bas"
6     'merge "LINIENSTAERKE.bas"
9
10    fullw 2: clearw 2
20    anfangsform=2 : endform=1 :gosub linienendform
30    for staerke = 21 to 1 step -4
40    gosub linienstaerke
50    y=20+staerke*12
60    linef 400,y,100,y
70    next
80    warte = inp(2)
90    end
```

Listing 8: Einbahnstraße

Bibliotheken

Die jeweils benötigten Unterprogramme können nach Bedarf zu einem Hauptprogramm zugeladen werden. Dies geschieht durch einfaches 'MERGE "PROGRAMMNAME.BAS"'. Wenn mehrere Routinen benötigt werden kann dies manchmal etwas lästig werden. Deshalb sollte man sich Unterprogrammbibliotheken anlegen, die nach Aufgabe und Wirkung gegliedert sind. So wäre eine TEXTBIBLIOTHEK und eine GRAFIKBIBLIOTHEK empfehlenswert, mit folgendem Inhalt:

Textbibliothek	Grafikbibliothek
Schriftart	Rechteck
Schriftgröße	Rechteckrund
Textausgabe	Linienmuster
Textwinkel	Linienendform
Grafikmodus	Linienstärke

Anmerkung

Bei eventuell fehlerhaftem Aufruf von VDI-Routinen kann es vorkommen, daß das Basic auch auf einen anschließenden korrekten Aufruf nicht mehr richtig reagiert. Wenn sich also zum Beispiel die Schriftart nicht mehr umschalten lassen sollte, hilft meistens ein 'QUIT' mit anschließendem Neustart des Basics. Das öftere Zwischenspeichern von Programmen ist deshalb auch hier empfehlenswert.

Kombination der einzelnen Routinen

Die VDI-Routinen können wie alle normalen Basicbefehle auch kombiniert werden. So kann man dicke Geraden mit abgerundeten Enden als auch große Kursivschrift erzeugen. Die angegebenen Beispielprogramme sollen eine Anregung zum Eigenexperiment sein, wobei diese 'neuen Basicbefehle' sicher in so manchem Programm ihre Anwendung finden werden. Ein Beispiel, in dem von einigen Routinen Gebrauch gemacht wird, zeigt Bild 9 (Listing 9).

Fortsetzung im nächsten Heft.

- weitere Grafikbefehle
- Die Maus in Basic

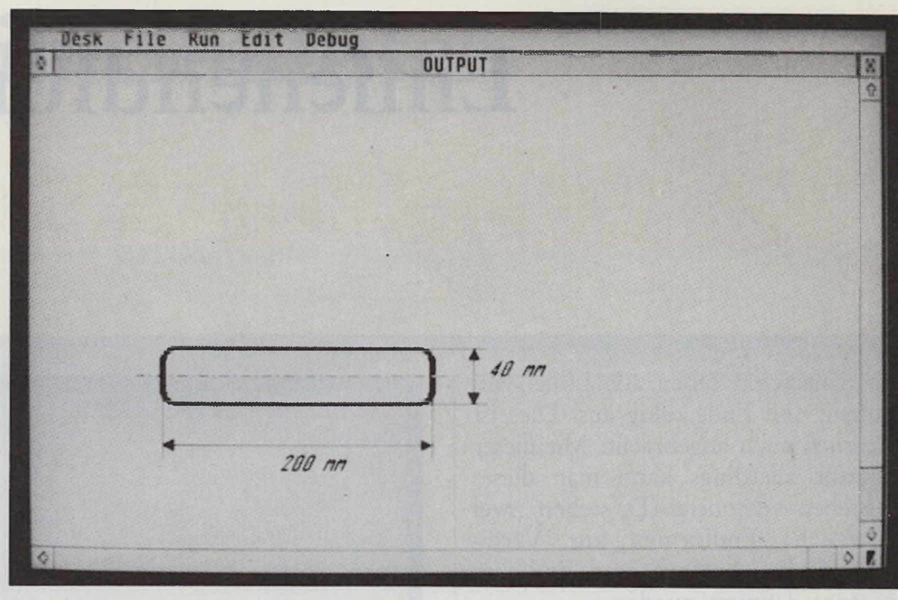


Bild 9

```
5       ' merge "LINIENSTAERKE.bas"
6       ' merge "LINIENENDFORM.bas"
7       ' merge "LINIENMUSTER.bas"
8       ' merge "SCHRIFTART.bas"
9       '
100     ' BEMASSUNG EINES RECHTECKES
110     fullw 2: clearw 2
120     staerke =3 : gosub linienstaerke
130     xpos1=100 :ypos1=200: xpos2=300  :ypos2=240
140     gosub rechteckrund
150     staerke =1 : gosub linienstaerke
160     linientyp=4 : gosub linienmuster
170     linef 80,220,320,220
180     linientyp=1 : gosub linienmuster
190     linef 100,230,100,280
200     linef 300,230,300,280
210     linef 295,240,340,240
220     linef 295,200,340,200
230     endform =1:anfangsform=1  :gosub linienendform
240     linef 330,200,330,240
250     linef 100,270,300,270
255     endform =0:anfangsform=0  :gosub linienendform
260     schrifttyp= 4: gosub schriftart
270     gotoxy 11,16 : print"200 mm"
280     gotoxy 20,12 : print"40 mm"
290     schrifttyp= 0: gosub schriftart
300     warte=inp(2)
310     end
```

Listing 9: Zeichnung

TURBO-PASCAL

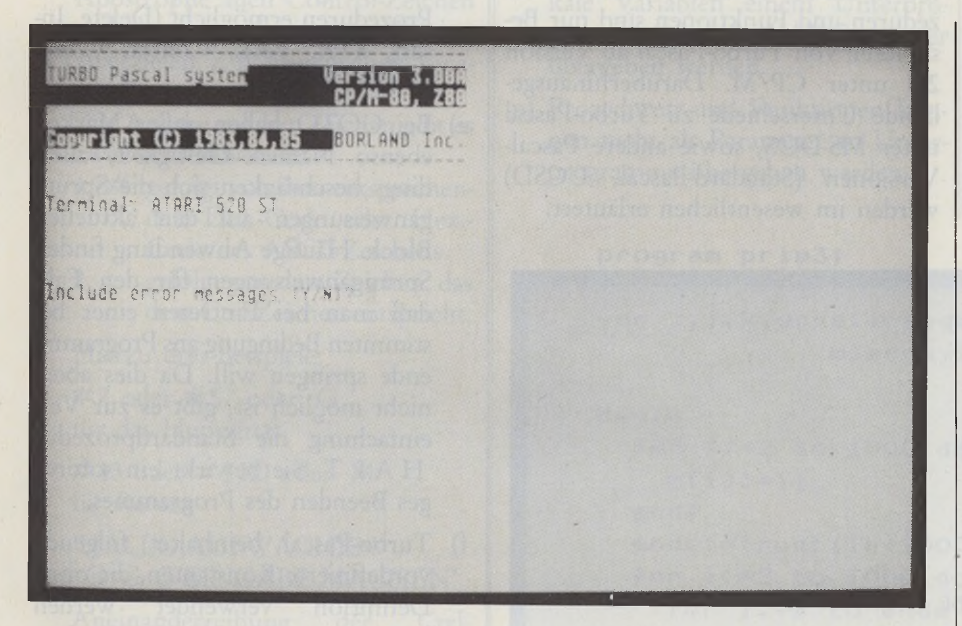


Bild 1: Einschaltmeldung

Standard-Pascal, von N. Wirth im Jahr 1970 eingeführt, fand aufgrund seiner Leistungsfähigkeit, der leichten Erlernbarkeit und vor allem wegen des strukturierten Aufbaus Verwendung in der Ausbildung. Aufgrund der weiten Verbreitung von Pascal entstand der Anspruch, diese Sprache auch auf den Ende der 70er Jahre aufkommenden Kleincomputern einzusetzen. So wurde eine auf diesen Anwendungsbereich bezogene Pascal-Version entwickelt, das sogenannte UCSD-Pascal. Während mit Standard-Pascal im wesentlichen mit Lochkarten gearbeitet wurde und Dateien auf Magnetbändern angelegt waren, hat USCD-Pascal einen erweiterten Befehlssatz, auch speziell zum unmittelbaren Zugriff auf Dateien, die auf Magnetplatten bzw. Disketten gespeichert sind. Aber ein noch gewichtigerer Unterschied ist, daß UCSD-Pascal ein vollständiges Betriebssystem ist, bestehend aus Compiler, Linker, Texteditor, Utilities, verschiedenen Assemblern und einem System zur Wartung von Dateien. Da Editieren, Compilieren und Binden getrennt durchgeführt werden mußte, war neben dem Zeitfaktor ein relativ großer Arbeitsspeicher notwendig. Mit der 1985 erschienenen Version von Turbo-Pascal 3.0 der Fa. Borland Inc. wurden mehrere Vorzüge miteinander vereint.

Sie ist mit etwa 200,– sehr preiswert (UCSD-P. ca. 800 – 1000 DM), ein wesentlicher Vorteil aber liegt bei dem geringen Speicherbedarf von ca. 34 KB auf der Diskette. Der Befehlssatz liegt nicht über dem der bisher bekannten Pascal-Versionen, hier wurde vielmehr auf eine Effizienz der Befehle wert gelegt, die sich in einer äußerst bequemen Handhabung bemerkbar macht.

Turbo-Pascal ist menügesteuert und hat keinen expliziten Compiler oder Binder wie andere Pascal-Versionen, das heißt, es ist nicht möglich, Teile eines Quelltextes getrennt zu übersetzen und bei Bedarf mit anderen Programmen zu binden. Es gibt aber die Möglichkeit vollständige Programme mit den Standard-Prozeduren Chain und Execute zu koppeln. Dies mag man als Nachteil sehen, der jedoch wegen der Schnelligkeit des Turbo-Pascal Compilers in Vergessenheit gerät. Durch diese Menüsteuerung ist Turbo-Pascal um bis zu <<<100>>>mal schneller als gewöhnliche Pascal-Versionen beim Erzeugen von lauffähigen Programmen.

Unter CP/M ruft man Turbo-Pascal mit "TURBO" auf. Darauf erscheint das Eingangsmenü und fragt, ob zu Fehlermeldungen ein Kommentar angezeigt werden soll (Bild 1).

Wenn mit Y geantwortet wird, werden bei auftretenden Fehlern nicht nur der Fehlercode sondern auch ein dementsprechender selbstklärender englischer Text ausgegeben. Nun wird das Hauptmenü angezeigt, das alle notwendigen Optionen zur Steuerung beinhaltet.

Hauptmenü

Hier nun eine kurze Erklärung der zur Verfügung stehenden Befehle:

LOGGED DRIVE:
Bezeichnet das zur Zeit angemeldete Laufwerk (das kann natürlich auch eine RAMDISK sein).

ACTIVE DIRECTORY:
Ist nur unter MS-DOS von Bedeutung. Dort kann man Unterverzeichnisse auf der Diskette anlegen.

WORK FILE:
Hier ist die zu bearbeitende Datei einzugeben, die, nachdem das Laufwerk angegeben wurde, geladen wird. Dasselbe gilt auch für das Neuanlegen einer Datei.

MAIN FILE:
Ist anzugeben wenn ein Programm über mehrere Dateien verteilt wurde.

COMPILE:
Compiliert ein Programm.
RUN:
Ausführung des Programms.
SAVE:
Sichert die Datei auf dem angemeldeten Laufwerk unter dem bereits angegebenen Namen (Work File).
DIR:
Zeigt den Inhalt des angemeldeten Laufwerkes.
QUIT:
Beendet Turbo-Pascal.
EXECUTE:
Lädt das angegebene Codefile (.COM) und führt es aus.

```
Logged drive: A

Work file: A:PRIM.PAS
Main file:

Edit     Compile  Run     Save

eXecute  Dir      Quit  compiler Options

Text:     0 bytes (8118-8118)
Free: 24813 bytes (8119-E206)

>
Dir mask: t*.*
TINST    COM : TINST    DTA : TINST    MSG : TINSTMSG OVR : TURBO    COM
TURBO    MSG : TURBO    OVR

Bytes Remaining On A: 144k
>
```

Bild 2: Hauptmenue

OPTIONs:
Hier können Compiler-Direktiven gewählt werden [Compilen im Speicher, Compilen auf Diskette (direkt aus CP/M aufrufbar)].
EDIT:
Ruft den bildschirmorientieren Texteditor, der auf Wordstar aufgebaut worden ist, auf. Dieser ist verständlicherweise nicht mit einem so umfangreichen Befehlssatz versehen wie Wordstar. Nähere Einzelheiten entnehme man dem Handbuch.

Turbo-Pascal ist momentan für vier Betriebssysteme erhältlich:

1. CP/M-80
2. CP/M-86
3. MS-DOS
4. PC-DOS

Dabei verfügen die Versionen für IBM und Kompatible noch über einige Grafikfunktionen. Diese fehlen den CP/M-Versionen gänzlich.

Die in diesem Bericht erwähnten Prozeduren und Funktionen sind nur Bestandteil von Turbo-Pascal ab Version 2.0 unter CP/M. Darüberhinausgehende Unterschiede zu Turbo-Pascal unter MS-DOS, sowie andere Pascal-Versionen (Standard-Pascal, UCSD) werden im wesentlichen erläutert.

Im folgenden wird auf die Besonderheiten von Turbo-Pascal aufmerksam gemacht, die ein besonders effizientes Arbeiten möglich machen:

a) Bei der Case-Anweisung ist ein Else erlaubt, das ausgeführt wird, wenn der Ausdruck nach Case einen Wert hat, der unter den Konstanten nicht vorkommt.
b) Die Standardprozeduren Write und Read wurden generell für alle Ein- und Ausgaberoutinen eingeführt, da diese eine schnellere Abarbeitung ermöglichen sowie erheblich vielseitiger und leichtverständlicher sind als GET und PUT. Hier sind keine Datenpuffervariablen nötig.
c) Ein Programm kann o h n e Programmkopf beginnen.
d) Der Datentyp PACKED und UNPACKED hat in Turbo-Pascal keine Wirkung, ist aber dennoch erlaubt, um Standard-Pascal Programme auch unter Turbo-Pascal lauffähig zu halten.
Daher gilt: Die Datentypen packed Array [] of Char und Array [] of Char werden wie der Datentyp String [] behandelt, der zudem noch Zugriff auf einige Standard-Prozeduren ermöglicht (Delete, Insert, STR, VAL, Concat, Copy, Length, Pos).
e) Bei GOTO haben außer Marken ebenso Namen Gültigkeit, allerdings beschränken sich die Sprunganweisungen auf den aktuellen Block. Häufige Anwendung finden Sprunganweisungen für den Fall, daß man bei Eintreten einer bestimmten Bedingung ans Programmende springen will. Da dies aberr nicht möglich ist, gibt es zur Vereinfachung die Standardprozedur H A L T. Sie bewirkt ein sofortiges Beenden des Programmes.
f) Turbo-Pascal beinhaltet folgende vordefinierte Konstanten, die ohne Definition verwendet werden können.

Name	Typ	Wert
Pi	Real	3.1415926536E+00
Maxint	Integer	32767
True	Boolean	Wahrheitswert true
False	Boolean	Wahrheitswert false

g) Neben den Konstanten gibt es auch die sogenannten Typenkonstanten. Diese sind eigentlich Variable, die im Konstantendefinitionsteil einen Anfangswert erhalten. Sie dürfen nicht in der VAriablendefinition vereinbart werden, und können neue Werte zugewiesen bekommen.
h) In Turbo-Pascal gibt es drei Varianten von Dateien: Allgemeine Dateien, Text-Dateien und typenlose Dateien. Letztere ermöglichen Zugriffe auf Datensätze von 128 Byte Länge, die sehr schnell und speichereffizient erfolgen. Die Prozeduren Read und Write werden durch Blockread und Blockwrite ersetzt, die jeweils 128 Byte Datensätze übertragen. Vorzugsweise zum Kopieren von Dateien eignet sich dieser Typ.

i) Die Angabe von Hexadezimalkonstanten ist zulässig, allerdings ohne Angabe von Vorzeichen.

DEZ 0...32767
entspricht HEX $0000...$7FFF

DEZ – 1... – 32768
entspricht HEX $FFFF...$8000

j) Zeichenketten, Konstanten des Typ Char, können außerhalb der Apostrophe auch Control-Zeichen enthalten, um bestimmte Effekte zu erzielen, z. B. Hupzeichen, Escape, Return.
Zur Angabe solcher Zeichen gibt es zwei Schreibweisen:
1. # gefolgt von dem entsprechenden Dez.-Code oder Hex.-Code des ASCII-Zeichens.
2. gefolgt von dem Zeichen, das dem Ctrl-Zeichen entspricht.

Hierzu drei Beispiele:

#7 oder #$7 oder G
für das Hupsignal

#13 oder #$0D oder M
für Return

'HALLO,AUFWACHEN'
G G G'EINGABE MACHEN'

Aneinanderreihung der Ctrl-Zeichen ohne Trennzeichen.

k) Es gibt die folgenden logischen Operatoren für Werte des Typs
1. INTEGER:
NOT,AND,OR,XOR,SHL,SHR
2. BOOLEAN:
NOT,AND,OR,XOR

l) Für Programme die die Kapazität des Arbeitsspeichers sprengen, gibt es die sogenannte Overlay-Technik. Die besagt, daß Teile des Objektcodes erst bei Bedarf in den Arbeitsspeicher geladen werden, so laufen Programme, die insgesamt nicht in den Speicher passen würden.

Durch den Vorsatz OVERLAY wird bewirkt, daß der Compiler den Objektcode nicht in der Datei des Hauptprogrammes ablegt, sondern in einer gesonderten Overlay-Datei, aus der dann die einzelnen Overlays nachgeladen werden. Overlays können ihrerseits wieder Overlays aufrufen (allerdings nicht sich selbst). Durch Overlays können nur Prozeduren und Funktionen gekennzeichnet. Sehr speichereffiziente Technik, aber relativ zeitaufwendig, wenn sich die Overlaydatei nicht auf RAM-Disk oder Festplatte befindet.

Weitere Informationen zum Overlay, insbesondere der Compiler Direktive zur Erzeugung der Overlaydatei, müssen dem Handbuch entnommen werden.

m) Bei rekursiven Aufrufen dürfen lokale Variablen einem Unterprogramm nicht als VAR-Parameter übergeben werden.

n) Prozeduren und Funktionen können nicht als Parameter an Unterprogramme übergeben werden.

```
program prim3;

var i,j,k,ende:integer;
        m:array[1..10000] of integer;

begin
  for i:=2 to 1000 do begin
    m[i]:=i;
  end;
  ende:=trunc(ln(1000))+1;
  for i:=2 to 1000 do begin
    for j:=2 to ende do begin
      k:=i*j;m[k]:=0;
    end;
  end;
  for k:=2 to 1000 do begin
    if m[k]<>0 then write(m[k],' ');
  end
```

Bild 3: Primzahlenprogramm

o) Die Standardprozedur PAGE, die einen Seitenvorschub bewirkt, ist unter CP/M nicht verfügbar.

p) Für dynamische Variablen werden die Prozeduren NEW, MARK und RELEASE verwendet. Sie bieten Kompatibilität zu anderen Pascal-Versionen.

Noch ein Wort zu den komfortablen Fehlermeldungen. Wie schon anfangs erwähnt, kann zusätzlich zu dem Fehlercode auch ein entsprechender Kommentar angezeigt werden. Damit erspart man sich ein häufiges Nachschlagen im Handbuch, besonders dann, wenn man mehrere Fehler in einem Programm hat, da der Compiler bereits beim ersten gefundenen Fehler abbricht. Das heißt, daß alle Fehler erst nach mehrmaligem Compilieren des Programmes lokalisiert werden können. Mittels der Escape-Taste kommt man bei Entdeckung des Fehlers automatisch in den Editier-Modus, in dem der Cursor schon unmittelbar an oder hinter dem Fehler steht. Durch den bequemen Übergang von Compiler in den Editier-Modus und des sehr schnellen Compilers kann man von keinem spürbaren Mangel sprechen.

Mit dem folgenden Pascal-Programm wurden der ATARI 520 ST mit Turbo-Pascal V3.0 unter CP/M und der Commodore PC10 mit Turbo-Pascal V2.0 unter MS-DOS verglichen.

Der Atari brauchte für das Compilieren und die Ausführung dieses Programmes, das die Primzahlen bis 1000 ausgibt, etwa 11 S., während der Commodore nur 3.5 S. benötigte. Diesen noch erheblichen Zeitunterschied wird man auf den Emulator zurückführen können, mit dem der CP/M Modus auf dem ATARI ST simuliert wird.

Wünschenswert wäre deshalb eine Turbo-Pascal Version speziell für 68000-Prozessoren. Eine auf den ATARI ST zugeschnittene Version könnte dann die Vorzüge der IBM-Version in Bezug auf Geschwindigkeit und Grafikmöglichkeiten noch übertreffen.

```
List of \DZGOUT.BAS

10    '------ Erstellen einer Direktzugriffsdatei ------
20    '
30    closew 3:clearw 2:fullw 2
40    dnr=1
50    open "R",#1,"Daten.dzg",24
60    field 1,20 as n$,4 as g$
70    gotoxy 1,1:print
80    print "Bitte Datendiskette einlegen und Taste drücken !"
90    a=inp(2)
100   print
110   input "Name : ";ns$
120   if ns$="*" then close 1:end
130   input "Geburtsjahr : ";g
140   lset n$=ns$
150   lset g$=mki$(g)
160   put #1,dnr
170   dnr=dnr+1
180   print
190   goto 110
```

Listing 2:
Erstellen einer Direktzugriffsdatei

griffsdateien sind allerdings mehr Programmschritte erforderlich als für eine sequentielle Datei, was somit einen größeren Programmieraufwand bedeutet. Außer dem Vorteil des wahlfreien Zugriffs kann man normalerweise mit Direktzugriffsdateien auch Platz auf der Diskette sparen. Das rührt daher, daß nicht, wie bei sequentiellen Dateien im ASCII-Format abgespeichert wird, sondern im Binär-Format. Aus diesem Grund kann man zum Beispiel im BASIC Maschinenprogramme in hexadezimaler Form in DATA-Zeilen erstellen, anschließend über den READ-Befehl in ein Variablenfeld einlesen und dann als lauffähiges Maschinenprogramm mit der OPEN-Anweisung in dem Modus „R" auf Diskette abspeichern. Dieses „R" werden vielleicht noch viele von Commodores C 64 her kennen, denn es ist das Kürzel für relativ und relative Dateien und Direktzugriffsdateien sind genau dasselbe. Zu den Datensätzen wäre noch zu sagen, daß sie eine Länge bis zu 32 767 Bytes haben können, also nicht von der Größe eines Sektors auf der Diskette (512 Bytes) abhängig sind. Daraus ergibt sich, daß ein Teil des Datensatzes in dem einen und der andere Teil in dem anderen Sektor stehen kann.

Neben den Befehlen CLOSE, OPEN, LOC und LOF, die ja bereits oben ausführlich beschrieben wurden, werden folgende Befehle bei Direktzugriffsdateien benutzt:

Die CVD-, CVI- und DVS-Befehle

Numerische Werte, die aus einer Direktzugriffsdatei gelesen wurden, müssen von Zeichenketten in Zahlen umgewandelt werden. Dies geschieht mit den CV-Befehlen. Dabei wandelt CVD eine acht Bytes umfassende Zeichenkette in eine Zahl doppelter Genauigkeit um. CVS wandelt eine Vier-Byte-Zeichenkette in eine Zahl einfacher Genauigkeit um und CVI wandelt eine Zwei-Byte-Zeichenkette in eine Integerzahl um. Die Funktionen CVD, CVI und CVS ändern dabei nicht die Bytes der aktuellen Daten, sondern sie ändern nur den Weg, wie BASIC diese interpretiert.

Der **FIELD**-Befehl

Mit dem FIELD-Befehl legt man Platz für Variablen in einem Puffer für Direktzugriffsdateien an. Dabei liest FIELD aber keine Daten in den Puffer. Man kann diesen Befehl also in etwa mit dem DIM-Befehl vergleichen, der ja auch nur Platz für Datenfelder schafft. Zu beachten ist noch, daß die Gesamtzahl der Bytes, die der FIELD-Anweisung zugeordnet werden, nicht die Datensatzlänge überschreitet, die

```
List of \DZGIN.BAS

10    '------ Zugriff auf eine Direktzugriffsdatei ------
20    '
30    closew 3:clearw 2:fullw 2
40    open "R",#1,"Daten.dzg",24
50    field 1,20 as n$,4 as g$
60    gotoxy 1,1:print
70    print "Bitte Datendiskette einlegen und Taste drücken !"
80    a=inp(2)
90    print
100   input "Datensatznummer (99=Ende) : ";dnr
110   if dnr=99 then close 1:end
120   get #1,dnr
130   print "Name : ";n$
140   print "Geburtsjahr : ";cvi(g$)
150   goto 90
```

Listing 3:
Lesen einer Direktzugriffsdatei

zuvor in der OPEN-Anweisung angegeben wurde, da man ansonsten eine Fehlermeldung bekommt.

Der **GET**-Befehl
Mit dem GET-Befehl liest man einen Datensatz aus einer Direktzugriffsdatei in einen Puffer. Da das BASIC und das TOS so viel wie möglich Datensätze in dem Puffer für Direktzugriffsdateien zwischenspeichern, liest die GET-Anweisung nicht unbedingt bei jedem Zugriff von Diskette, da die Daten auch schon im Puffer stehen können.

Der **LSET**- und der **RSET**-Befehl
Mit diesen Befehlen werden die Daten in den Datenpuffer für Direktzugriffsdateien geschrieben. Hinter ihnen muß eine Zeichenkettenvariable stehen, die zuvor in der FIELD-Anweisung definiert wurde. Wenn diese Variable weniger Bytes benötigt als in der FIELD-Anweisung angegeben, werden die restlichen Bytes mit Leerstellen aufgefüllt. Bei LSET wird die Variable linksbündig und bei RSET rechtsbündig abgespeichert. Falls die Variable mehr Bytes hat als zuvor definiert, werden die Zeichen rechtsbündig abgeschnitten. Numerische Werte müssen zuvor in Zeichenketten umgewandelt werden (Siehe nächsten Befehl).

Die **MKD$**-, **MKI$**- und **MKS$**-Befehle
Jeder numerische Wert, der mit den LET- und RSET-Befehlen in den Puffer für Direktzugriffsdateien gebracht wird, muß zuvor in eine Zeichenkette umgewandelt werden. Dies geschieht durch die Befehle MKD$, MKI$ und MKS$. Sie bewirken das genaue Gegenteil der CVD-, CVI- und CVS-Befehle. Die MK-Befehle unterscheiden sich von dem STR$-Befehl dadurch, daß sie nicht die Datenbytes ändern und BASIC sie anders interpretiert.

Der **PUT**-Befehl
Mit dem PUT-Befehl werden die Daten aus dem Puffer für Direktzugriffsdateien auf Diskette geschrieben. Zuvor müssen die Daten mit PRINT #, PRINT# USING, WRITE#, LSET oder RSET in den Puffer gebracht werden. Im Falle von WRITE # wird der Puffer vom BASIC mit Leerstellen bis zum Wagenrücklauf bzw. Zeilenvorschub aufgefüllt. Jeder Versuch über das Ende des Puffers hinauszuschreiben oder zu lesen, ergibt eine Fehlermeldung. Wie beim GET-Befehl werden die Daten solange im Puffer zwischengespeichert bis er voll ist und erst dann auf Diskette geschrieben. Bei einem Reset würden also alle Daten, die im Puffer stehen, verloren gehen.

Erstellen einer Direktzugriffsdatei

Wie bei der sequentiellen Datei soll nun in der folgenden Anleitung gezeigt werden, wie man eine Direktzugriffsdatei erstellt.

1. Zuerst muß die Datei mit dem OPEN-Befehl in dem Modus „R" geöffnet werden. Dabei muß die Datensatzlänge in dem OPEN-Befehl bestimmt werden. Wird die Datensatzlänge weggelassen, wird als Standardlänge 128 Bytes angenommen.
2. Mit dem FIELD-Befehl wird dem Puffer Platz für die Variablen zugeteilt, die auf Diskette geschrieben werden sollen.
3. Mit LSET oder RSET werden die Daten in den Puffer gebracht. Zuvor müssen die Daten mittels der MK-Befehle in Zeichenketten umgewandelt werden.
4. Die Daten werden nun mit dem PUT-Befehl auf die Diskette geschrieben.

Zugriff auf eine Direktzugriffsdatei

Wenn man sich jetzt eine Direktzugriffsdatei erstellt hat, will man die Daten ja auch wieder in den Speicher lesen können. Dazu gibt die folgende Anleitung die Möglichkeit.

1. Die Datei muß zuerst einmal wieder mit dem OPEN-Befehl im Modus „R" geöffnet werden.
2. Ebenfalls muß der FIELD-Befehl wieder für Platz im Puffer sorgen. Allerdings braucht man, falls man in demselben Programm ein- und ausliest, den OPEN- und den FIELD-Befehl nur einmal benutzen. Die Datei muß also nicht wie bei der sequentiellen Datei zuerst geschlossen und dann wieder geöffnet werden.
3. Mit der GET-Anweisung wird nun der gewünschte Datensatz von der Diskette in den Puffer gelesen.
4. Die Daten können jetzt vom Programm verarbeitet werden. Numerische Werte müssen allerdings, wie schon mehrfach erwähnt, zuerst mit den CV-Befehlen umgewandelt werden.

Listing 2 und 3 geben ein Beispiel für Direktzugriffsdateien

Die Kombination aus Sequentieller- und Direktzugriffsdatei

Eine einfache und überaus sinnvolle Methode Dateien zu verwalten besteht darin, die beiden bisher kennengelernten Dateienarten miteinander zu koppeln. Da man davon ausgehen kann, daß Daten so gut wie immer im Arbeitsspeicher sortiert werden, ist somit zweckmäßig alle Daten, die sortiert werden sollen, in eben diesen Speicher einzulesen. Dafür bietet sich eine sequentielle Datei an. In dieser sequentiellen Datei muß nun neben den betreffenden Daten auch noch die zu den Daten passende Datensatznummer stehen. In der dazugehörigen Direktzugriffsdatei befinden sich, nach Datensätzen geordnet, die übrigen Daten, die nicht sortiert werden sollen.

Wenn man nun zum Beispiel nach einem bestimmten Datensatz sucht, dann lädt man die sequentielle Datei in ein Datenfeld im Speicher. Dort läßt man jetzt nach dem ausgewählten Kriterium suchen. Nachdem man es gefunden hat, spaltet man mit einem Stringbefehl (LEFT$, RIGHT$ oder MID$) von dem sequentiellen Datensatz die Datensatznummer für die Direktzugriffsdatei ab. Damit hat man jetzt die Möglichkeit die restlichen Daten aus der Direktzugriffsdatei nachzuladen. Man spart somit eine Menge Platz im Arbeitsspeicher, da der vollständige Datensatz in zwei Teile gespalten worden ist und nur ein Teil davon sich im Speicher befindet. Diese Methode hat aber auch wiederum den Nachteil, daß man nur nach bestimmten Daten suchen kann, nicht nach allen.

Ein Beispiel für diese Art von Datenverarbeitung kann man an Listing 4 sehen.

```
List of \KOMBIDAT.BAS

10   '------ Kombinierte Datei ------
20   '
30   closew 3:clearw 2:fullw 2
40   dim ns$(20),nl$(20)
50   schreiben:
60   x=1
70   open "O",#1,"Seqkombi.dat"
80   open "R",#2,"DzgKombi.dat",16
90   field 2,4 as g$,12 as t$
100  gotoxy 1:print
110  input "Name (max.8 Zeichen) : ";ns$(x)
120  if ns$(x)="*" then goto 210
130  if len(ns$(x))<8 then ns$(x)=ns$(x)+" ":goto 130
140  ns$(x)=ns$(x)+str$(x)
150  input "Geburtsjahr : ";g
160  input "Tel.-Nr. (max.12 Zeichen) : ";tel$
170  gosub direktschreiben
180  x=x+1
190  print
200  goto 110
210  print
220  print "Bitte Datendiskette einlegen und Taste drücken !"
230  a=inp(2)
240  for i=1 to x-1
250  print#1,ns$(i)
260  next i
270  close 1:print
280  goto lesen
290  '
300  direktschreiben:
310  lset g$=mki$(g)
320  lset t$=tel$
330  put #2,x
340  return
350  '
360  '---------------------------------------------------------
370  '
380  lesen:
390  x=1
400  open "I",#1,"Seqkombi.dat"
410  if eof(1) then close 1:goto 460
420  input#1,nl$(x)
430  x=x+1
440  goto 410
450  print
460  suchen:
470  input "Welchen Namen wollen Sie lesen  : ";ni$
480  if len(ni$)<8 then ni$=ni$+" ":goto 480
490  for i=1 to x-1
500  nv$(i)=left$(nl$(i),8)
510  if ni$=nv$(i) then goto 550
520  next i
530  print "Name nicht vorhanden !"
540  goto 460
550  gosub direktlesen
560  ausgeben:
570  print "Name : ";ni$
580  print "Geburtsjahr : ";cvi(g$)
590  print "Tel.-Nr.: ";t$
600  print
610  input "Noch einen Datensatz (j/n) ";jn$
620  if jn$="j" then goto suchen
630  if jn$="n" then close:end
640  if jn$<>"j" or jn$<>"n" then goto 610
650  direktlesen:
660  dnr=val(right$(nl$(i),2))
670  get #2,dnr
680  return
```

Listing 4:
Kombinierte Datenverarbeitung

Druckertreiber

für Itoh, NEC u. a. Drucker

Wer einen ITOH 8510 oder NEC 8023 oder dazu kompatiblen Drucker besitzt, kann die im Betriebssystem des ATARI implementierte Hardcopy-Routine, die über das gleichzeitige Drücken der <Alternate> und der <Help> Taste aufgerufen wird, nicht nutzen. Außerdem kann auch nicht ohne Programmhilfe der komplette ATARI-Zeichensatz (früher IBM-kompatible genannt) ausgedruckt werden, also Umlaute und Sonderzeichen, die über die <Alternate>-Taste des ATARI 260ST / 520ST / 520ST+ anzuspechen sind.

Das Programm installiert die neue Hardcopy-Routine in der VBL-Queue (Vertical-**BL**ank-Interrupt), die bei angeschlossenem monochrom Monitor 71mal in der Sekunde aufgerufen wird, und es dem Benutzer ermöglicht eigene Programme abarbeiten zu lassen. Bei Drücken von ALT/HELP wird dann die neue Hardcopy-Routine ausgeführt und durch erneutes Drücken von ATL/HELP abgebrochen. Um die Hardcopy-Routine an andere Drucker anzupassen, müssen in den 'Datas: Druckersteuerung' die entsprechenden Escape-Sequenzen verändert werden. Die erste Zahl gibt jeweils die Anzahl der zu übertragenden Daten minus 1 an.

'dinit:'
Druckerinitialisierung / wird nur zu Beginn der Hardcopy gesendet.

'dzeile:'
wird bei neuer Bildschirmzeile (640 Punkte) gesendet.

'dreset:'
Druckerreset / wird am Ende der Hardcopy gesendet.

Außerdem ist zu beachten, daß die Einzelnadelansteuerung beim ITOH-Drucker von oben nach unten erfolgt, während diese bei anderen Druckern oft umgekehrt aufgebaut ist. In diesem Fall muß der Assembler-Befehl **clr d4** (über 'loops:') durch **move #560,d4** und **add #80,d4** (zwei Befehle unter 'loops:') durch **sub #80,d4** ersetzt werden.

Aufbau und Werte der Einzelnadelansteuerung im Graphikmodus:

ITOH	Nadeln	z.B. EPSON
1	1. ★	128
2	2. ★	64
4	3. ★	32
8	4. ★	16
16	5. ★	8
32	6. ★	4
64	7. ★	2
128	8. ★	1

Das in das Programm integrierte Druckertreiber wandelt den ATARI-Code in eine Codesequenz für den Drucker um, die das entsprechende Zeichen oder ein durch Einzelnadelgraphik definiertes Symbol darstellen. So besteht die Möglichkeit, daß Sie sich Ihre eigenen Zeichen und Symbole in der Tabelle der Wandlungswerte definieren. Der Treiber kann durch folgendes Assemblerprogramm erzeugt werden.

Das Programm kann nur vom Desk-Top des ATARI aus initialisiert werden, nicht von einem „Auto"-Ordner aus. Außerdem darf es nur einmal gestartet werden.

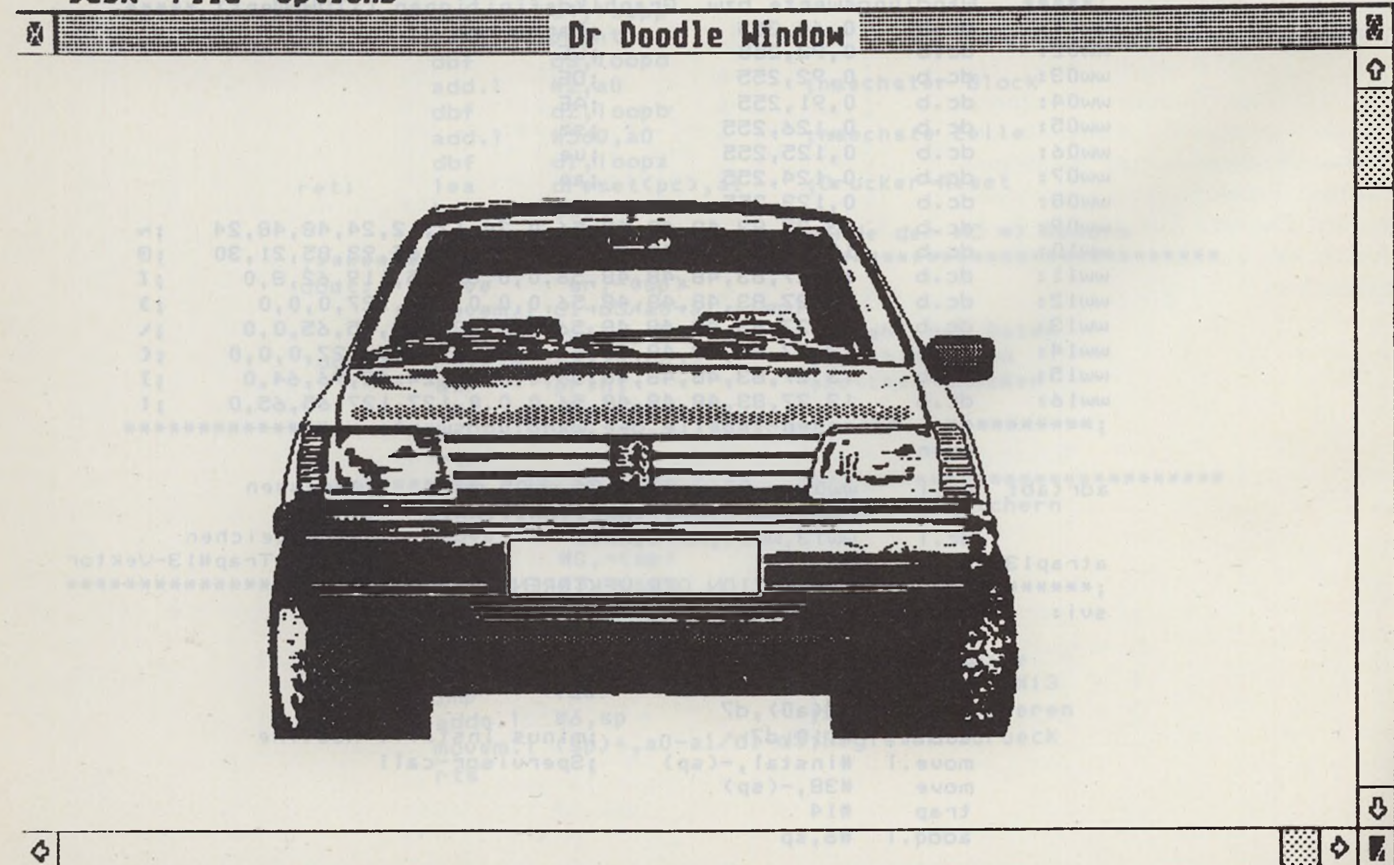
```
          Sonderzeichen auf Itoh - Drucker 8510 A
                     und Compatiblen

                 Zeichen und ASCII-Wert
ä = 132  ,  ö = 148  ,  ü = 129  ,  ß = 158
Ä = 142  ,  Ö = 153  ,  Ü = 154  ,  § = 221
[ = 125  ,  ] = 092  ,  { = 093   ,  } = 124
@ = 064  ,  \ = 123  ,  ~ = 126  ,  | = 091
-----------------------------------------------
```

```
;****************************************************************
;*          BILDSCHIRM - HARD - COPY  und                       *
;*       DRUCKERTREIBER  fuer ITOH 8510 A,                      *
;*          NEC 8023 B und Compatible                           *
;****************************************************************
nvbl:    equ    $454          ;Anzahl der VBL-Slots
vbl:     equ    $456          ;Basisadr. des VBL
ahf:     equ    $4ee          ;ALT-HELP-Flag
badr:    equ    $44e          ;Bildschirm-Basis-Adr.
;****************************************************************
         jmp    svi           ;Sprung zur Installation
;*********   Test - Alt/Help - Routine ************************
begin:   tst    ahf
         bne    return
         move   #0,ahf
         bsr    hc
         move   #-1,ahf
return:  rts                  ;Return zu Aufrufer
;*********   Hard - Copy - Routine  *************************
hc:      lea    dinit(pc),a1  ;Drucker initialisieren
         bsr    dout
         move.l badr,a0       ;Videobasis in a0
         move   #49,d1        ;50 * 8 Bit (200 Punkte)
loopz:   tst    ahf           ;Hardcopy abbrechen ?
         bne    ret
         lea    dzeile(pc),a1 ;neue Zeile
         bsr    dout
         move   #3,d2         ;40 * 16 Bits (640 Punkte)
loopb:   move.l #buffer,a1    ;neuer Block
         move   #15,d3
         clr    d4
loops:   subq   #1,d3         ;Block in Buffer
         move   0(a0,d4),0(a1,d3)
         add    #80,d4
         dbf    d3,loops
         move   #15,d3        ;Block drucken
loopd:   move   #7,d4
         move.l #buffer,a1
loopp:   lsl    (a1)+         ;Zeichenausgabe
         roxl   #1,d0
         dbf    d4,loopp
         bsr    print
         dbf    d3,loopd
         add.l  #2,a0         ;naechster Block
         dbf    d2,loopb
         add.l  #560,a0       ;naechste Zeile
         dbf    d1,loopz
ret:     lea    dreset(pc),a1 ;Drucker-Reset
         bsr    dout
         rts                  ;Ende der HC => Return
;*********   Data - out - Routine   ************************
dout:    move   sr,-(sp)
         movem.l d1-d5/a0-a1,-(sp)
         move.b (a1)+,d5      ;Anzahl der Daten
loopo:   move.b (a1)+,d0      ;Zeichen holen
         bsr    print         ;Zeichen drucken
         dbf    d5,loopo
         movem.l (sp)+,a0-a1/d1-d5
         rtr
;***********   Print - Routine   *****************************
print:   movem.l d1-d5/a0-a1,-(sp);Register sichern
         move   d0,-(sp)
         move   #0,-(sp)
         move   #3,-(sp)
         pea    radr(pc)      ;Returnadresse
         move   sr,-(sp)
print1:  move.l atrap13,a0    ;Zeichenausgabe
         jmp    (a0)          ;=> alten trap#13
radr:    addq.l #6,sp         ;Stack Korregieren
         movem.l (sp)+,a0-a1/d1-d5;Register zurueck
         rts
```

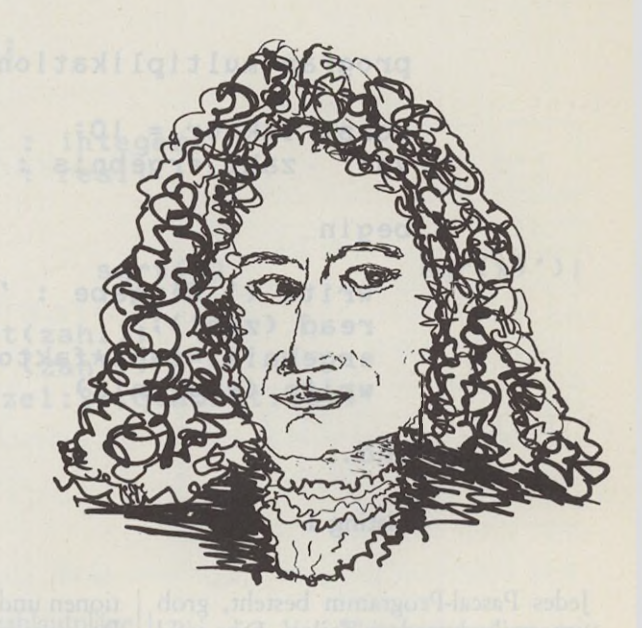
```
;*********   Datas: Druckersteuerung  ***********************
            even
   dinit:   dc.b     7,27,">",27,"N",27,"T16"       ;unidire.,(N),16/144
   dzeile:  dc.b     7,13,10,27,"S0640"             ;CR,Graphikp.(640)
   dreset:  dc.b     7,27,"<",27,"A",13,10,10,10 ;bidire.,1/6 Zeilenvorschub
            even
   buffer:  blk.w    8
   ;**********   Neue trap#13 Routine     ***********************
   trap13:  move.l   sp,a2                 ;Stackp. sichern
            btst     #5,(sp)               ;Supervisor call
            bne      prints                ;wenn ja, verzweige
            move.l   usp,a2                ;sonst usp benutzen
            subq     #6,a2
   prints:  cmp      #3,6(a2)              ;Zeichenausgabe-Aufruf ?
            bne      print1
            cmp      #0,8(a2)              ;zum Drucker ?
            bne      print1
            move     10(a2),d0             ;Zeichen holen
            lea      zeitab(pc),a0         ;Zeiger auf Sonderzeichentabelle
            moveq    #ww01-zeitab-1,d1;Laenge der Tabele
   loop:    cmp.b    (a0)+,d0              ;Zeichen mit Tabelle vergleichen
            beq      printi                ;wenn ja => ITOH-CODE drucken
            dbra     d1,loop               ;wenn d1>-1 dann Spung zu loop
            bra      print1                ;existiert nicht, so drucken
   printi:  lsl      #2,d1
            lea      adrtab(pc),a0         ;Zeiger auf Adresstabelle
            move.l   (a0,d1),a0            ;Zeiger auf Tabelle
            move.b   (a0)+,d1              ;Laenge der Tabelle -1
   loop1:   move.b   (a0)+,d0              ;Zeichen aus Tabelle
            cmp      #255,d0               ;vergleiche Endekennung
            beq      exit                  ;wenn d1=255 dann Sprung zu exit
            bsr      print                 ;Zeichenausgabe
            dbf      d1,loop1              ;naechstes Zeichen
   exit:    rte                            ;zurueck zum Programm
   ;**********   Sonderzeichen-Tabelle (ATARI-Wert)  ***************
   zeitab:  dc.b     091,092,093,123    ;| ] { \
            dc.b     124,125,064,126    ;} [ @ ~
            dc.b     132,148,129,158    ;ae oe ue sz
            dc.b     142,153,154,221    ;AE OE UE Pararaph
   ;*****    Wandlungswerte bzw. Graphikdefinitionen (ITOH-Wert) *****
   ww01:    dc.b     0,64,255           ;Paragraph
   ww02:    dc.b     0,93,255           ;UE
   ww03:    dc.b     0,92,255           ;OE
   ww04:    dc.b     0,91,255           ;AE
   ww05:    dc.b     0,126,255          ;sz
   ww06:    dc.b     0,125,255          ;ue
   ww07:    dc.b     0,124,255          ;ae
   ww08:    dc.b     0,123,255          ;ae
   ww09:    dc.b     13,27,83,48,48,48,56,0,24,12,12,24,48,48,24     ;~
   ww10:    dc.b     13,27,83,48,48,48,56,0,62,65,65,93,85,21,30     ;@
   ww11:    dc.b     13,27,83,48,48,48,56,0,0,65,65,119,62,8,0       ;[
   ww12:    dc.b     13,27,83,48,48,48,56,0,0,0,127,127,0,0,0        ;}
   ww13:    dc.b     13,27,83,48,48,48,56,0,8,62,119,65,65,0,0       ;\
   ww14:    dc.b     13,27,83,48,48,48,56,0,65,65,127,127,0,0,0      ;{
   ww15:    dc.b     13,27,83,48,48,48,56,0,6,12,24,48,96,64,0       ;]
   ww16:    dc.b     13,27,83,48,48,48,56,0,0,0,127,127,65,65,0      ;|
   ;**********   Adressen-Tabelle der Wandlungswerte    *************
            even
   adrtab:  dc.l     ww01,ww02,ww03,ww04,ww05,ww06     ;Adressen
            dc.l     ww07,ww08,ww09,ww10,ww11,ww12     ;  der
            dc.l     ww13,ww14,ww15,ww16               ;ITOH-Zeichen
   atrap13: dc.l     0                                 ;alter Trap#13-Vektor
   ;***********  INSTALLATION DER VEKTOREN  ***************************
   svi:     move.l   4(sp),a0              ;Programmlaenge
            move.l   #$100,d7
            add.l    12(a0),d7
            add.l    20(a0),d7
            add.l    28(a0),d7
            sub.l    #110,d7                         ;minus Install.-Routine
            move.l   #instal,-(sp)         ;Spervisor-call
            move     #38,-(sp)
            trap     #14
            addq.l   #6,sp
```

```
;*********  INITIALISIERUNG DES trap#13-VEKTORS ****************
        move.l   #trap13,-(sp)        ;neuer Vektor
        move     #45,-(sp)            ;Vektornummer
        move     #5,-(sp)
        trap     #13                  ;Vektor setzen
        addq.l   #8,sp
        move.l   d0,atrap13           ;alten Vektor sichern
        move.l   #0,-(sp)
        move.l   d7,-(sp)             ;Speicherplatz des
        move     #$31,-(sp)           ;Programms reservieren
        trap     #1                   ;=> Desktop
;*********  INSTALLATION DES ALT-HELP-VEKTORS  *****************
instal: move     nvbl,d0              ;Anzahl der Slots
        lsl      #2,d0
        move.l   vbl,a0               ;Basisadr. der Slots
        clr      d1
search: tst.l    0(a0,d1)             ;freien Slot suchen
        beq      found
        addq     #4,d1
        cmp      d0,d1
        bne      search
        rts
found:  lea      (a0,d1),a2           ;freie Slotadr. merken
        move.l   #begin,(a2)          ;Vektor setzen
        rts
;*********   ENDE  DER  INSTALLATION  **********************
```



Beispiel für eine Hardcopy mit dem Itoh-Drucker

Einführung in die Programmiersprache Pascal



EINFÜHRUNG IN PASCAL

Pascal ist neben Fortran und Basic eine der meistverbreitetsten höheren Programmiersprachen. Sie findet ihe Anwendung vor allem im mathematisch-naturwissenschaftlichen Bereich, ist aber beispielsweise ebenso für die Datenverwaltung geeignet. Das Hauptmerkmal dieser Sprache liegt in ihrem übersichtlichen Aufbau, der durch spezielle Befehle ermöglicht wird. Dazu gehören verschiedene Anweisungen zur Schleifensteuerung:

WHILE DO,
REPEAT UNTIL,
FOR NEXT

sowie komfortable Verzweigungsmöglichkeiten:

IF THEN ELSE,
CASE OF

Dies bildet die Grundlage für eine strukturierte Programmierweise, der Pascal seine hohe Popularität verdankt. Ein Pascalprogramm besteht aus sogenannten Blöcken bzw. Modulen. Diese besitzen nur einen Eingang und auch nur einen Ausgang, wodurch unkontrollierte und schwer überschaubare Sprünge innerhalb eines Programmes vermieden werden. Desweiteren resultiert aus diesem modularen Aufbau eine gute Lesbarkeit der Programme für den Programmierer selbst und auch für den Anwender. Bei vielen anderen Programmiersprachen wurden strukturierte Befehle nachträglich übernommen. So zum Beispiel bei FORTRAN 77 und bei verschiedenen BASIC-Versionen. Auch das Basic des ATARI ST verfügt über solche Befehlserweiterungen gegenüber dem 'Standard Basic' (WHILE ... WEND, IF ... THEN ... ELSE).

Bleibt die Frage, für welchen Pascal Compiler man sich entscheiden sollte. Wer bisher nur mit Interpretersprachen (BASIC, LOGO) gearbeitet hat, wird bei einem Compiler mit einigen Nachteilen konfrontiert werden.

Nach dem Editieren eines Programmes benötigt jeder Compiler eine gewisse Zeit um ein ablauffähiges Programm zu erzeugen. Man muß hierbei unbedingt unterscheiden zwischen den Diskettencompilern und Compilern, die komplett innerhalb des Rechenspeichers arbeiten. Erstgenannte benötigen meist hohe Compilierzeiten und sind in ihrer Handhabung recht umständlich. Dies ist allerdings eine Eigenschaft mit der alle Compilersprachen zu kämpfen haben, sei es nun FORTRAN, C oder andere.

Eine sehr erfreuliche Ausnahme stellt TURBO-PASCAL dar. Es handelt sich hierbei um einen Compiler, der samt Editor ständig im Rechnerspeicher vorhanden ist. Ein Pascalprogramm kann innerhalb des Rechners übersetzt werden und erreicht somit eine hohe Compiliergeschwindigkeit, die zumindest bei kürzeren Programmen stark an einen Interpreter erinnert. Turbo-Pascal ist das ideale Werkzeug für Einsteiger und Fortgeschrittene. Über den CP/M-Emulator ist Turbo-Pascal bereits für den ATARI ST verfügbar. (siehe Bericht in dieser Ausgabe).

Doch nun zu der Serie **'Einführung in PASCAL'**, die in diesem Heft beginnt. Sie soll einen Einblick in die strukturierte Programmierung und die damit gegebenen Vorteile dieser und darauf aufbauender Programmiersprachen (C, MODULA-2) geben. Pascal bietet dafür gute Voraussetzungen, weil man nach einer kurzen Eingewöhnungszeit gut mit den Besonderheiten und Möglichkeiten zurechtkommt. In diesem Kurs wird deshalb zuerst ein Überblick über diese Möglichkeiten gegeben. Dabei wird nicht alphabetisch jeder Befehl abgehandelt, denn dies kann man im Handbuch nachlesen. Es soll vielmehr anhand von Programmbeispielen gezeigt werden, wie man übersichtliche Programme erstellt und wie man Problemstellungen in eleganter Weise löst.

Die Programme werden mit Turbo-Pascal Version 3.0 erstellt, sie sind jedoch auf allen gängigen Pascal-Compilern lauffähig solange keine Einschränkungen angegeben werden.

Der Kurs beginnt mit einem einfachen Beispiel, an dem man jedoch schon wichtige Merkmale von Pascal aufzeigen kann. Geben Sie nun das Programm (Listing 1) ein und compilieren Sie es. Sollten dabei Fehler auftreten, so müssen Sie das Quellprogramm mit dem Editor verbessern. Wenn das Programm fehlerfdrei übersetzt wurde, können Sie es starten. Geben Sie nun eine beliebige Zahl ein und drücken Sie RETURN. Als Ergebnis erscheint das zehnfache Ihrer Zahl auf dem Bildschirm.

```
program multiplikation;

const faktor = 10;
var   zahl,ergebnis : real;

begin

  write (' Eingabe : ');
  read (zahl);
  ergebnis:=zahl*faktor;
  write (ergebnis)

end.
```

Listing 1

Jedes Pascal-Programm besteht, grob unterteilt, aus drei Teilen. Dies sind der Programmkopf, der Vereinbarungsteil und der Anweisungsteil. Der **Programmkopf** ist in vielen Pascalversionen zwingend notwendig. Bei Turbo-Pascal kann er jedoch weggelassen werden. Es empfiehlt sich aber trotzdem einen Programmnamen anzugeben, weil man damit den Inhalt des Listings beschreiben kann. Als Programmkopf wurde bei diesem Beispiel 'program multiplikation' gewählt.

Achtung, bei manchen Pascal-Versionen darf der Programmname nur acht Zeichen lang sein!

In nun folgenden **Vereinbarungsteil** werden alle Labels (Sprungzeichen), Konstanten, Typen, Variablen, Funktionen und Prozeduren vereinbart. Der Programmierer muß also vor Beginn des Programms festlegen, welche Konstanten, Variablen usw. er im Programm verwendet und von welchem Datentyp sie sind. Bei diesem Programm wird eine **Konstante** (CONST) mit dem Namen 'faktor' definiert, und ihr wird der Wert 10 zugewiesen. Hiernach folgt die Vereinbarung der Variablen (VAR). Die Reihenfolge der Vereinbarungen ist bei Pascal **nicht** beliebig. Die Konstantenvereinbarung muß vor der Vereinbarung der Variablen stehen. Eine komplette Übersicht der Prioritäten im Vereinbarungsteil erfolgt später. Als Variablen werden 'zahl' und 'ergebnis' festgelegt. ihr **Datentyp** steht hinter einem Doppelpunkt und ist der Real-Typ. Dies bedeutet, daß die **Variablen** Gleichtkommwerte annehmen können.

Bei der Benennung der **Variablen** hat man viele Möglichkeiten. Am Anfang darf nur keine Zahl stehen und Leerzeichen können nicht verwendet werden. Es empfiehlt sich aussagekräftige Namen zu verwenden und nicht nur einzelne Buchstaben. Allerdings ist nur eine bestimmte Anzahl von Zeichen relevant. Bei Turbo-Pascal sind es acht. Dies bedeutet, daß das Programm zwischen 'ergebnis' und 'ergebniswert' nicht unterscheiden kann. Dies kann zu schwer auffindbaren Fehlern im Programm führen. Mit dieser Anweisung ist der Vereinbarungsteil abgeschlossen. Es dürfen in dem nun folgenden Programm also nur die definierten Variablen und die Konstante 'faktor' vorkommen, alle anderen würden zu einer Fehlermeldung bei der Compilierung führen.

Der **Anweisungsteil** wird durch die Befehle:

begin und **end.**

begrenzt, wobei der Punkt das Ende des gesamten Programms anzeigt. Diese Befehle umschließen immer eine Anweisung oder einen Anweisungsblock und finden somit in Pascal-Programmen öfter ihre Anwendung. Der Anweisungsblock beginnt hier mit einer **'write'**-Anweisung, die den in Hochkommas stehenden Text auf dem Bildschirm ausgibt. Die darauffolgende **'read'**-Anweisung wartet auf eine Eingabe des Benutzers, die mit <RETURN> abgeschlossen werden muß. Die eingegebene Zahl, alphanumerische Zeichen sind nicht erlaubt, wird der Variablen 'zahl' übergeben. In der nächsten Zeile wird der Variablen 'ergebnis' das Produkt aus 'zahl' und 'faktor' zugewiesen. Die **Zuweisung** hat in Pascal die Form:

Variable : = numerischer Ausdruck

Dabei muß aber auf den Datentyp geachtet werden. Man kann zwar eine Integerzahl einer Realvariablen zuweisen, aber nicht umgekehrt. Wenn man also einer Integervariablen eine Realzahl zuweisen will, bricht das Programm schon beim Compilieren mit einer Fehlermeldung ab. Mit dem 'write'-Befehl wird nun die Variable 'ergebnis' auf dem Bildschirm ausgegeben. Das Programm ist damit beendet, für einen erneuten Durchlauf muß man es wieder starten. Nach diesem Durchlauf werden Sie feststellen, daß die Ausgabe auf dem Bildschirm unübersichtlich ist, weil alle Ausgaben zusammenhängen. Sie können deshalb die Befehle 'write' und 'read' durch **'writeln'** und **'readln'** ersetzen. Da-

Programmkopf

Vereinbarungsteil

Anweisungsteil

Bild 1 Programmaufbau

```
program multiplikation;const faktor=10;var zahl,ergebnis:real;begin
write(' Eingabe : ');read (zahl);ergebnis:=zahl*faktor;write (ergebnis);end
```

Listing 2

durch steht jede Ausgabe in einer neuen Zeile. Auffällig bei einem Pascal-Programm ist, daß alle Anweisungen durch ein **Semikolon** getrennt werden müssen. Von dieser Regel gibt es nur wenige Ausnahmen. So muß zum Beispiel hinter 'begin' kein Semikolon stehen, und auch hinter der letzten Anweisung vor einem 'end' ist ein Trennzeichen nicht notwendig. Man kann sie aber trotzdem setzen, sogar mehrere hintereinander haben keine negativen Auswirkungen. Dagegen darf vor einem 'else' kein Semikolon stehen und auch hinter die letzte 'end'-Anweisung darf keines.

Das Beispielprogramm enthält auch einige **Leerzeilen** und **Leerzeichen**. Diese sind nicht notwendig. Man kann sogar mehrere Anweisungen in eine Zeile schreiben (siehe Listing 2), doch würde das der Übersichtlichkeit des Programms sehr schaden.

In Pascal ist es also möglich, beliebige Abstände zwischen Befehlen zu setzen und somit ein Listing optisch zu strukturieren. Auch die Großschrift kann dazu verwendet werden. Sie kann äquivalent zu Kleinbuchstaben verwendet werden und kann mit diesen, sogar innerhalb eines Wortes, gemischt werden.

Dieses Programm wird Zeile für Zeile ausgeführt, es läuft also von oben nach unten durch. Diese Programmart nennt man **Folgestruktur**. Diese Struktur läßt sich einfach als **Struktogramm** (auch Strukturdiagramm oder Nassi-Shneiderman-Diagramm) darstellen (siehe Bild 2). Mit diesen Diagrammen werden alle grundlegenden Programmstrukturen erklärt. Sie eigenen sich sehr gut für die strukturierte Programmierung und haben die früher

```
program potenzen;

label 10;
var zahl,quadrat : integer;
         wurzel : real;

  begin
    writeln ('          sqrt(x)          sqr(x)');
10: read (zahl);
    wurzel :=sqrt(zahl);
    quadrat:=sqr (zahl);
    writeln (wurzel:20,quadrat:10);
    goto 10;
 end.
```

Listing 3

verwendeten Programmablaufpläne (PAP) ersetzt.

Das zweite Programm (Listing 3) verwendet wieder einige Elemente, die auch im ersten Programm vorkamen. Zusätzlich wird hier mit einem **Label** gearbeitet. Ein Label ist eine Sprungmarke, die das Ziel eines 'GOTO'-Sprunges markiert. Diese Möglichkeit des Sprunges entspricht dem 'GOTO'-Befehl in Basic und sollte in Programmen möglichst vermieden werden, weil er das Listing unübersichtlich macht. Labels müssen vor der Verwendung definiert werden. In einer Zeile können mehrere Labels vereinbart werden (z. B. label 10,20,1). In Turbo-Pascal dürfen, im Gegensatz zu Standard-Pascal, auch Namen als Labels definiert werden.

Ein Sprung darf jedoch nicht in eine strukturierte Anweisung (Schleife, Auswahlblock) erfolgen! Außerdem sind berechnete Sprünge nicht möglich!

Bei der Variablenvereinbarung tritt der Datentyp **Integer** auf. Er legt die Variablenzahl und -quadrat als Ganzzahlen fest. Dies beschränkt ihren Wertebereich auf −32768 bis 32767.

Die Funktionen **'sqrt(x)'** und **'sqr(x)'** müssen genau von der Basic-Funktion 'sqr' unterschieden werden. 'sqr(x)' bedeutet in Pascal das Quadrat (square) von x. Das Gegenteil ist dann 'sqrt(x)', die Quadratwurzel (squareroot). Das Argument der beiden Funktionen ist in diesem Fall vom Datentyp 'integer', es kann jedoch auch als 'real' definiert werden. Wenn das Argument der Funktion 'sqr' als Integer festgelegt wird, ist auch das Ergebnis im Integerbereich. Das bedeutet, daß es nicht größer als 32767 werden kann. Bei höherem Argument ist dann das Ergebnis falsch, es wird negativ! Ist das Argument als Real-Zahl festgelegt, so treten diese Probleme nicht auf. Bei der Funktion 'sqrt' darf das Ergebnis

Konstante : faktor = 10 Variablen : zahl, ergebnis
Eingabe : zahl
ergebnis := zahl * faktor
Ausgabe : ergebnis

Bild 2 Folgestruktur

	Argument	Ergebnis
sqr (x)	real, integer	real, integer*
sqrt (x)	real, integer	real

* nur bis 32767 (!)

Bild 3 Funktion sqr(x) und sqrt(x)

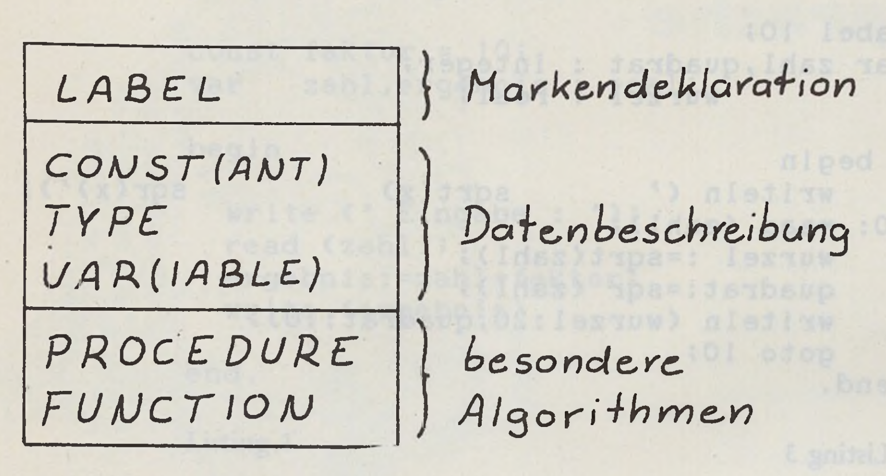


Bild 4: Reihenfolge beim Vereinbarungsteil

nicht vom Typ Integer sein (siehe auch Bild 3)!

Der Befehl 'writeln' wurde schon zuvor angesprochen, er wird hier jedoch noch für die **formatierte Ausgabe** erweitert. Da eine Ausgabe der Form 'write (x,y)' die Werte von x und y **ohne** Zwischenraum hintereinander schreibt, wird eine andere Form des 'write'-Befehls benutzt. Er hat die Form:

write (wurzel:20,quadrat:10)

Hinter jeder Variablen wird nach einem Doppelpunkt die Länge des reservierten Zeilenplatzes angegeben. Die Variable 'wurzel' hat also einen Platz, der von der letzten Cursorposition zwanzig Stellen nach rechts reicht. In diese Spalten wird nun ihr Wert **rechtsbündig** hineingeschrieben. Der Platz für die nächste Variable beginnt dann an der nächsten Stelle und reicht zehn Plätze nach rechts. Wenn der reservierte Platz zu klein ist, wird er bei der Ausgabe überschrieben. Es entsteht dann ein unübersichtliches Zahlengewirr. Deshalb sollte man das Format der Ausgabe gut durchdenken und eventuell mehr Platz als benötigt reservieren. Bei Dezimalzahlen kann man die Anzahl der '**Nachkommastellen**' festlegen. Der Befehl hat dann die Form:

write (x:10:6)

Dabei gibt die erste Zahl wieder die Gesamtlänge des reservierten Platzes an. Die zweite Zahl die darin enthaltenen Nachkommastellen. Ansonsten gilt dasselbe wie für die Integer Zahlen. Anstelle der Zahlen können auch Variablen oder Konstanten stehen:

write (x:10:i)

Nachdem nun bei diesem Beispiel auch mit einem Label gearbeitet wurde, ist es an der Zeit die Reihenfolge im **Vereinbarungsteil** darzustellen. Als erstes kommt die Labelvereinbarung, dann die der Konstanten, Typen, Variablen, Funktionen und zuletzt die der Prozeduren (siehe auch Bild 4). Funktionen, Prozeduren und Typen werden später im Text erklärt. Beim Vereinbaren muß man darauf achten, daß Namen bzw. Bezeichnungen nicht mehrmals verwendet werden. Die Compilierung wird dann mit 'Duplicate identifier or lable' abgebrochen.

Das dritte Programm (Listing 4) enthält als Kern eine **einfache Auswahlanweisung**. Sie wird in Bild 5 als Struktogramm dargestellt. Mit der 'if'-Anweisung wird kontrolliert, ob die eingegebene Zahl Eins ist. Dann wird der Text 'hallo' ausgegeben. Das Programm geht dann in der nächsten Zeile weiter und gibt den Text 'wie geht es?' aus. Falls die eingegebene Zahl Zwei ist, überspringt das Programm die Zeile, die hinter der '**then**'-Anweisung steht und gibt nur den Text 'wie geht es ?' aus. Da es hierbei nur eine Möglichkeite gibt, nennt man diese Art der Auswahl auch **einseitige Auswahl**.

Variablen : eingabe
Eingabe : eingabe
eingabe = 1 ?
ja | nein
Ausgabe: 'hallo'
Ausgabe: 'wie geht es ?'

Bild 5 einfache Auswahlstruktur

```
program auswahl;

var eingabe : real;

begin
  writeln ('1 oder 2');
  readln (eingabe);
  if eingabe = 1
     then write ('hallo ');
  writeln ('wie geht es ?')
end.
```

Listing 4

```
program QuadratischeGleichung;

var x1,x2,p,q,wurz : real;

begin
    writeln ('Quadratische Gleichung');
    writeln ('gib p und q ein !');
    readln (p,q);
    wurz:=(p*p/4-q);
    if wurz<0
      then writeln ('keine relle loesung !')
    else if wurz=0
      then writeln ('nur ein loesung :',-p/2)
    else
      begin
        wurz:=sqrt(wurz);
        x1:=-p/2+wurz;
        x2:=-p/2-wurz;
        writeln ('x1 =',x1);
        writeln ('x2 =',x2);
      end;
   writeln ('--------------------------')
end.
```

Listing 5

Variablen : X1, X2, p, q, wurz

Eingabe : p, q

wurz < 0 — ja / nein

ja: Ausgabe: 'keine reelle Lösung'

nein: wurz = 0 — ja / nein

ja: Ausgabe: 'nur eine Lösung' -p/2

nein: wurz := √wurz; X1 := -p/2 + wurz; X2 := -p/2 - wurz; Ausgabe: X1, X2

Ausgabe : '----------'

Bild 6: mehrseitige Auswahlstruktur

Eine größere Bedeutung hat jedoch die mehrseitige Auswahl, die in Bild 6 dargestellt wird. Mit diesem Programm (Listing 5) wird eine quadratische Gleichung mit der p-q-Formel gelöst. Die Variable 'wurzel' bekommt den Inhalt des Wurzelausdrucks zugewiesen. Nun wird mit **'if'** abgefragt, ob der Wurzelausdruck kleiner als Null ist. Das Ergebnis wäre dann negativ und die Wurzel läßt sich nicht berechnen. Deshalb wird der Text 'keine reelle Lösung' ausgedruckt. Danach wird mit der Anweisung fortgefahren, die hinter dem Auswahlblock steht. Wenn nun aber die Bedingung (wurz<0) nicht erfüllt ist, so wird bei **'else'** nachgesehen, ob diese Bedingung (wurz=0) zutrifft. Gegebenenfalls wird dann 'nur eine Lösung' ausgegeben. Treffen beide Bedingungen nicht zu, so ist die zweite '**else**'-Abfrage an der Reihe. Hier muß keine Bedingung gestellt werden, weil alle anderen Fälle bereits herausgesiebt wurden. Bei der '**if-then-else**'-Anweisung wird also nur eine der Bedingungen ausgeführt und dann mit der Anweisung, die hinter dem Abfrageblock steht, fortgefahren (siehe Struktogramm). Dabei können vor dem letzten 'else' noch beliebig viele Abfragen der Form **'else if...'** eingefügt werden.

Wie man aus dem Struktogramm (Bild 6) erkennen kann handelt es sich bei diesem Programm um einen Sonderfall der zweiseitigen Auswahl, weil mehrere bzw. zwei Auswahlmöglichkeiten ineinander verschachtelt sind. Wenn man jedoch die Zeile mit der 'else if'-Anweisung streicht erhält lman eine reine zweiseitige Auswahl.

Zu Beachten ist hierbei, daß vor einem **'else' kein Semikolon** stehen darf! Soll hinter einer Abfrage mehr als eine Anweisung stehen, so **muß** diese mit **'begin ... end'** umschlossen werden!

(Fortsetzung folgt)

(MN)

Einkaufsführer

BENSE KG



Computer-Systeme Software-Entwicklung
☎ 0 25 41/52 31
Dieselstraße 10-12
D-4420 Coesfeld 1

COMPUTER ■ SHOP ■
Am EKZ Kupferpassage
Ausgang Gartenhof
☎ 0 25 41/7 23 59
Süringstraße 7
D-4420 Coesfeld 1

OCB

Wallstraße 3
4422 Ahaus

Heinicke-Electronic

Kommenderiestr. 120 · 4500 Osnabrück
Telefon 05 41 - 8 27 99

Wir liefern Micro-Computer seit 1978

Horten
COMPUTERCENTER

Wittekindstr. 23
4500 Osnabrück
✆ 05 41 - 2 85 43

Atari, Genie, Schneider, Tandy, Brother, Star, Memorex, BASF, Verbatim
cc Computer Studio GmbH
Software-Hardware-Beratung
Service-Eilversand

Ihre Ansprechpartner:
v. Schablinski
Jan P. Schneider

Elisabethstraße 5
4600 Dortmund 1
T. 0231/528184 Tx 822631 cccsd

G. Knupe GmbH

Postfach 354
4600 Dortmund 1

City Elektronic

Güntherstraße 75
4600 Dortmund
Telefon 02 31/57 22 84

GET

GESELLSCHAFT FÜR ELEKTRONISCHE TELEKOMMUNIKATION

IM SCHILDERN 15
4790 PADERBORN

TEL. (0 52 51) 2 60 41
BTX ★51051#

MICROTEC

Ges. für Microcomputer-Vertrieb mbH

Paul-Schwarze-Str. 5
4800 Bielefeld 14



CSF

CSF COMPUTER & SOFTWARE GMBH
Heeper Straße 106 – 108
4800 Bielefeld 1
Tel. (05 21) 6 16 63

---

computer store

D. Büschkamp
Schulstraße 9
D-4830 Gütersloh 1
Phone: 0 52 41 / 1 20 80

BÜROMASCHINEN
braun
AM RUDOLFPLATZ GmbH

5000 KÖLN 1
RICHARD-WAGNER-STR. 39
RUF: 02 21 / 21 91 71



EDV-Beratung · Organisation
Programmierung · Home/Personal-Computer
Software · Zubehör · Fachliteratur

Zeppelinstr. 7 · 5010 Bergheim
Telefon 0 22 71 - 6 20 96

Computer Center

Buchholzstraße 1
5060 Bergisch-Gladbach
Telefon 0 22 02 - 3 50 53

Rolf Rocke

Computer-Fachgeschäft
Austraße 1
5090 Leverkusen 3
Telefon 0 21 71 / 26 24

Computer Center

Luisenstraße 26
5200 Siegburg
Telefon 0 22 41/6 68 54

"BYTE ME" COMPUTERSYSTEME

Wilhelmstraße 7
D 5240 BETZDORF (SIEG)
Telefon (0 27 41) 2 35 37 u. 2 31 07

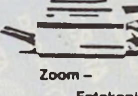
DR. AUMANN GMBH
Computer-Systeme

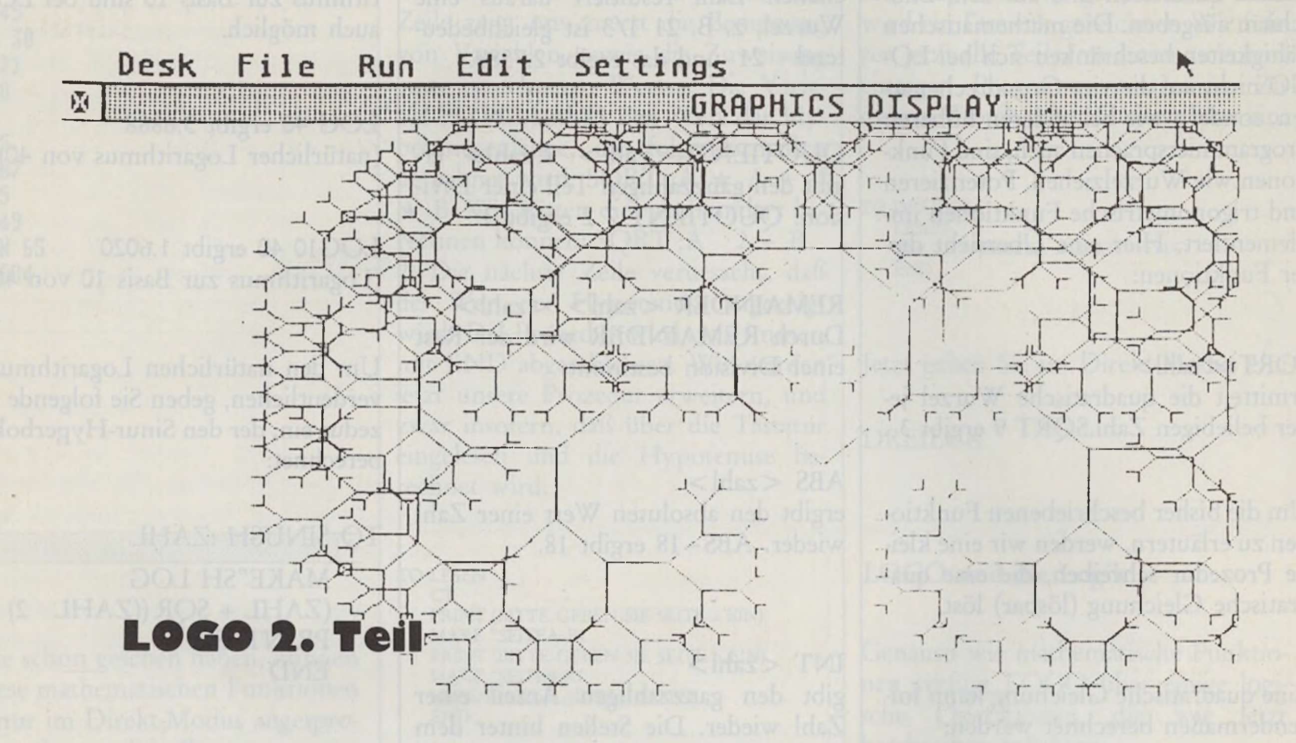
Schulstr. 12
5457 Straßenhaus
Telefon 0 26 34 - 40 81/2

City Elektronic

Güntherstraße 75
4600 Dortmund
Telefon 02 31/57 22 84







LOGO 2. Teil

– oder eine Einführung in Turtle-Geometrie

Wir haben uns in dem ersten Teil unseres LOGO-Kurses mit den einfachen Befehlen der Graphikerzeugung beschäftigt. Diesesmal verabschieden wir uns von der Welt der graphischen Darstellung um andere Seiten von LOGO kennenzulernen, kommen aber später wieder darauf zurück.

Das LOGO Einmaleins

Was hätte man von einer Computersprache, die keine Zahlen verarbeiten könnte? Schließlich können Computer hauptsächlich nur mit Zahlen umgehen. Genauso ist dies bei LOGO.

LOGO kennt eine große Anzahl von mathematischen Funktionen, die sowohl im Direkt-Modus angesprochen, als auch in Programme eingebaut werden können.

Die Addition zweier oder mehrerer Zahlen funktioniert folgendermaßen:

? + <zahl> <zahl>
oder
? <zahl> + <zahl>
wie
? SUM <zahl> <zahl>

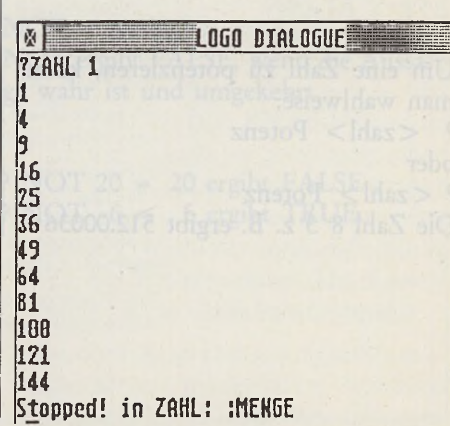
Diese drei Modi wiederholen sich bei Subtraktion, Multiplikationen, sowie Divisionen.

```
Desk  File  Run  Edit  Settings
LOGO DIALOGUE
DR LOGO FOR GEM!
?+ 12 34
46
?- 67 20
47
?65 * 2
130
?* 45 4
180
?/ 34 4
8.5
?36 / 3
12
?
```

Für die Benutzung von Produkten werden wir jetzt ein kleines Programm schreiben, das eine bestimmte Menge von Zahlen quadriert und auf dem Bildschirm auflistet. Tippen Sie bitte ein

```
TO ZAHL :MENGE
   IF :MENGE >100 (STOP)
   PRINT :MENGE ★ :MENGE
   ZAHL :MENGE + 1
   END
```

Jetzt geben Sie im Direct Modus ein:
ZAHL 1



Das Programm wird die ersten 100 Zahlen quadrieren und auf dem Bildschirm ausgeben. Die mathematischen Fähigkeiten beschränken sich bei LOGO nicht auf die vier Grundrechenarten, sondern wie bei anderen höheren Programmiersprachen auch, sind Funktionen wie Wurzelziehen, Potenzieren und trigonometrische Funktionen implementiert. Hier eine Übersicht dieser Funktionen:

SORT <zahl>
ermittelt die quadratische Wurzel jeder beliebigen Zahl.SQRT 9 ergibt 3 .

Um die bisher beschriebenen Funktionen zu erläutern, werden wir eine kleine Prozedur schreiben, die eine quadratische Gleichung (lösbar) löst.

Eine quadratische Gleichung kann folgendermaßen berechnet werden:

−b + SQRT b 2 − 4ac / 2a
−b − SQRT b 2 − 4ac / 2a

Unsere LOGO Prozedur sieht so aus:

```
TO QUADRATISCHE :A :B :C
CT
MAKE"ZWISCHEN SQRT B   2 – (4 ★ (:A ★ : C)
MAKE"QG1 :–(B + :ZWISCHEN) / 2 ★ :A
MAKE"QG2 :–(B – :ZWISCHEN) / 2 ★ :A
PRINT :QG1
PRINT :QG2
END
```

Jetzt geben Sie im Direkt-Modus ein:
QUADRATISCHE 9 12 24

Um eine Zahl zu potenzieren, kann man wahlweise:
? <zahl> Potenz
oder
? <zahl> Potenz
Die Zahl 8 3 z. B. ergibt 512.00036

Bei der Potenzierung mit einer gebrochenen Zahl resultiert daraus eine Wurzel, z. B. 21 1/3 ist gleichbedeutend 21 und das ergibt 2.7589.

QUOTIENT <zahl> <zahl> ergibt den ganzzahligen Teil einer Division. QUOTIENT 7 2 ergibt 3.

REMAINDER <zahl> <zahl>
Durch REMAINDER wird der Rest einer Division bestimmt.

ABS <zahl>
ergibt den absoluten Wert einer Zahl wieder. ABS−18 ergibt 18.

INT <zahl>
gibt den ganzzahligen Anteil einer Zahl wieder. Die Stellen hinter dem Dezimalpunkt werden ignoriert. INT 3.5 ergibt 3.

ROUND <zahl>
rundet eine Zahl auf bzw. ab. ROUND 20.5 ergibt 21. Außerdem kennt LOGO die Zahl PI, wobei PRINT PI die Zahl 3.141592 ergibt.

```
Desk  File  Run  Edit  Settings
LOGO DIALOGUE
?QUOTIENT 16 5
3
?REMAINDER 5 2
1
?ABS -9
9
?INT 16.3
16
?ROUND 13.5
14
?PI
3.141592
?
```

Die Exponentialfunktion einer Zahl ist durch EXP <zahl> zu ermitteln.

Natürlicher Logarithmus, wie Logarithmus zur Basis 10 sind bei LOGO auch möglich.

LOG 40 ergibt 3.6888
(natürlicher Logarithmus von 40)

LOG10 40 ergibt 1.6020
(Logarithmus zur Basis 10 von 40)

Um den natürlichen Logarithmus zu verdeutlichen, geben Sie folgende Prozedur ein, der den Sinur-Hyperbolikus berechnet:

```
TO SINUSH :ZAHL
    MAKE"SH LOG
    (ZAHL + SQR ((ZAHL   2) + 1)
    PRINT :SH
    END
```

Im Direct-Modus geben Sie ein:
SINUSH 20

Umwandlung von Bogenmaß in Altgrad, und umgekehrt beherrscht LOGO ebenfalls.

DEGREES <zahl> Umwandlung einer Zahl von Bogenmaß in Altgrad.

RADIANZ <zahl> Umwandlung einer Zahl von Altgrad in Bogenmaß.

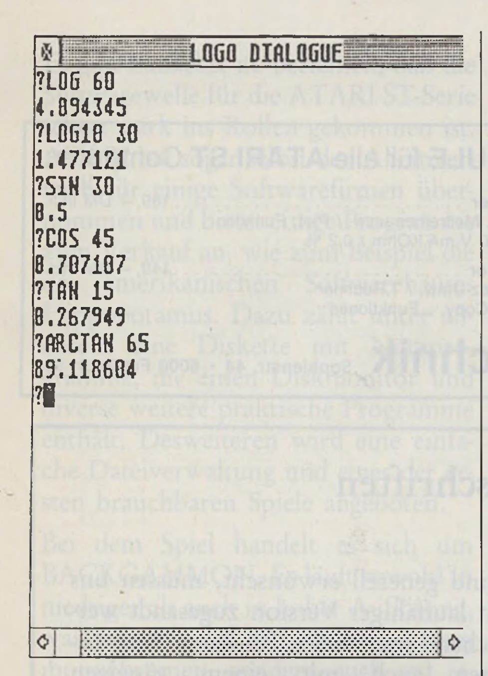
Die wichtigen trigonometrischen Funktionen sind auch vertreten.

SIN <zahl> ermittelt den Sinus eines Winkels.

COS <zahl> bestimmt den Cosinus eines Winkels.

TAN <zahl> ermittelt den Tangens eines Winkels.

ARCTAN <zahl> errechnet den Arcus-Tangens eines Winkels.



Wie Sie schon gesehen haben, können alle diese mathematischen Funktionen nicht nur im Direkt-Modus angesprochen, sondern auch in Programme eingebaut werden. Wir werden jetzt eine aufwendige Prozedur schreiben, in der Teile dieser Funktionen verdeutlicht werden sollen. Von Pythagoras wissen wir, daß die Länge der Hypotenuse eines rechtwinkligen Dreiecks folgendermaßen berechnet werden kann:

(Hypotenuse) 2=(a) 2+(b) 2

Eine Prozedur in LOGO, die die Hypotenuse berechnet, kann so aussehen:

```
TO HYPOTENUSE :SEITEA :SEITEB
  MAKE "HYPO SQRT :SEITEA ★ :SEITEA +
                  :SEITEB ★ :SEITEB
  PRINT :HYPO
  END
```

Wir wissen aus dem ersten Teil unseres Kurses, daß eine Prozedur mit den Befehl TO anfangen und einen Namen besitzen muß. Bei unserem Programm lautet der Name Hypotenuse. Die Prozedur läßt die Eingabe von zwei Parametern (a,b) zu. Die nächste Zeile zeigt uns zuerst die Benutzung von Variablen, sowie die Zuweisung eines errechneten Wertes. Die Variabel HYPO enthält den Wert der Hypotenusenlänge, die folgendermaßen berechnet wird: SQRT :A ★ A + B ★ B. Wir hätten dies auch anders berechnen können: SQRT :A 2 + B 2. Die nächste Zeile verursacht, daß der Wert der Hypotenuse angezeigt wird. Die Prozedur wird, wie immer, mit END abgeschlossen. Wir werden jetzt unsere Prozedur erweitern, und zwar insofern, daß über die Tastatur eingelesen und die Hypotenuse berechnet wird.

```
TO LESEN
CT
PRINT (BITTE GEBEN SIE SEITE a EIN)
MAKE "SEITEA RL
PRINT (BITTE GEBEN SIE SEITE b EIN)
MAKE "SEITEB RL
HYPOTENUSE :SEITEA :SEITEB
END
```

Zuerst wird durch CT das Textfenster freigestellt. Die nächste Zeile fordert Sie auf einen Wert über die Tastatur einzugeben. Durch RL (READ LIST) wird solange gewartet, bis eine Zahl eingegeben und mit RETURN abgeschlossen wird. Das Ganze wiederholt sich in der dritten und vierten Zeile damit dann in der fünften unsere alte Prozedur HYPOTENUSE aufgerufen werden kann. Unser Dreieck kann nicht nur berechnet werden, sondern auch gezeichnet. Das ist unser nächster Schritt. Geben Sie bitte ein:

```
TO ZEICHEN :SEITEA :SEITEB : HYPO
CS
FD :SEITEA
RT 90
FD :SEITEB
LT 180 - ARCTAN :A / :B
FD :HYPO
END
```

Wie oben werden auch hier durch CS die Graphikfenster gelöscht und dann wird ein Dreieck gezeichnet. Wir führen jetzt alle Teilprozeduren zu einem einzigen Programm zusammen. Zu diesem Zweck geben Sie jetzt bitte ein:

```
TO DREIECK
  LESEN
  ZEICHEN :SEITEA :SEITEB : HYPO
  END
```

Jetzt geben Sie im Direkt-Modus ein:

DREIECK

LOGO und die Logik

Genauso wie mathematische Funktionen verfügt LOGO über einige logische Operationen, die wir jetzt beschreiben möchten:

AND <Aussage> <Aussage>
Logisches UND. Ergibt TRUE, wenn alle Aussagen wahr sind, sonst ist die Antwort FALSE.

? AND 10 > 5 7 < 9 ergibt TRUE.
? AND 8 > 4 6 < 5 ergibt FALSE.

OR <Aussage> <Aussage>
Logisches ODER. Ergibt TRUE, wenn zumindest eine Aussage wahr ist.

? OR 10 > 5 6 < 5 ergibt TRUE.
? OR 8 > 9 6 < 5 ergibt FALSE.

NOT <Aussage>
NOT ergibt FALSE, wenn die Aussage wahr ist und umgekehrt.

? NOT 20 = 20 ergibt FALSE.
? NOT 6 < 5 ergibt TRUE.

```
Desk  File  Run  Edit  Settings
LOGO DIALOGUE
?AND 10 > 9 2 = 2
TRUE
?AND 2 = 3 8 < 9
FALSE
?OR 3 > 4 2 = 2
TRUE
?OR 10 < 5 4 > 7
FALSE
?NOT 10 < 5
TRUE
?
```

Damit ist unsere Folge zu einem Ende gekommen. Wir sind ein Stück weiter in das Reich von LOGO vorgedrungen. In unserer nächsten Folge werden wir uns intensiv mit Variabeln, sowie Listen beschäftigen.

Leserzuschriften

Dokumentierte Programmlistings sind generell erwünscht, müssen uns aber als Ausdruck und Diskette mit lauffähiger Version zugesandt werden. Disketten werden gegen ausreichend frankierten Rückumschlag zurückgesendet. Die Texte können auch mit einem gängigen Textprogramm (ST Text, 1st Word) erstellt werden und in Diskettenform zugesandt werden. Bei Veröffentlichung wird selbstverständlich ein angemessenes Honorar gezahlt.

Bauanleitungen, Softwaretests oder sonstige Berichte, Erfahrungen oder Tips sind gerne willkommen. Alle Fragen und Zuschriften richten Sie bitte nur schriftlich an

Uwe Bärtels
ST Redaktion
Postfach 1131
6242 Kronberg

Da es sich um ein privates Postfach handelt, bitte unbedingt den oben genannten Namen mit angeben.

Es gibt zunächst zu berichten, daß die Softwarewelle für die ATARI ST-Serie schon stark ins Rollen gekommen ist. ATARI hat sogar selbst den Alleinvertrieb für einige Softwarefirmen übernommen und bietet einige Programme zum Verkauf an, wie zum Beispiel die des amerikanischen Softwarehauses Hippopotamus. Dazu zählt unter anderem eine Diskette mit Nutzprogramme, die einen Diskmonitor und diverse weitere praktische Programme enthält. Desweiteren wird eine einfache Dateiverwaltung und eines der ersten brauchbaren Spiele angeboten.

Bei dem Spiel handelt es sich um BACKGAMMON. Es läuft sowohl in niedriger als auch in hoher Auflösung, was bedeutet, daß es nicht nur auf dem monochromen, sondern auch auf einem Farbmonitor läuft. Es lassen sich verschiedene Spielstärken einstellen, so daß auch Anfänger eine Chance gegen den Rechner haben.

Für alle die nicht so viel Geld für Software ausgeben wollen, gibt es inzwischen schon eine ganze Menge der sogenannten **Public-Domain-Software**. Dies ist wie schon im Editorial dieser Ausgabe erwähnt, Software die **kostenlos** auf Disketten kopiert werden darf. Sie verbreitet sich dann im „Schneeballsystem" in diesem unserem Lande. Den Löwenanteil an diesen Programmen stiftet allerdings ATARI selber, wie folgende Tabelle dieser ATARI-Programme zeigt:

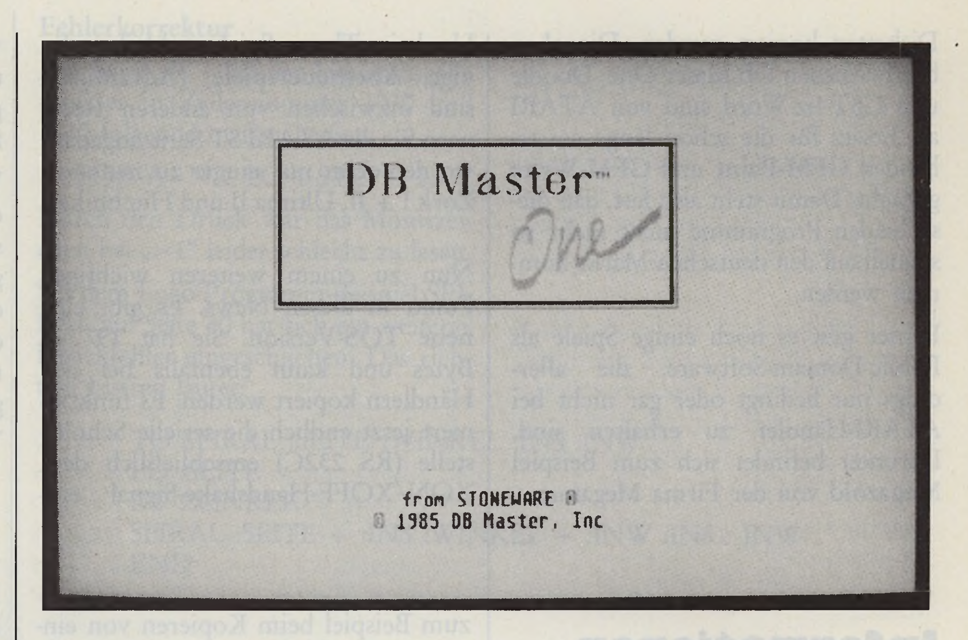


Bild 1

1. **DB Master One** – (siehe Bild 1) Dies ist eine komfortable Dateiverwaltung mit frei definierbarer Maske. Näheres darüber in der nächsten Ausgabe.
2. **GST – 1st – Word** – (siehe Bild 2) Ein Textverarbeitungsprogramm, das voll die GEM-Möglichkeiten, wie z. B. verschiedene Schriftarten auf dem Bildschirm ausnützt. Eine Anleitung wird auf der Diskette auf Englisch mitgeliefert. Auch dieses Programm wird in der nächsten Ausgabe näher besprochen.
3. **CP/M 2.2 Emulator** – Über dieses Programm wurde schon in der letzten Ausgabe berichtet. Zusätzlich dazu wurde nun auch noch eine Diskette mit CP/M-Nutzprogrammen herausgegeben. Eine Anleitung wird derzeit bei ATARI erstellt. Es sei noch erwähnt, daß die CP/M-Directory nur mit geladenem Emulator gelesen werden kann. Im normalen TOS erscheint als Meldung „0 Blocks belegt".
4. **Doodle** – Dies ist ein Zeichenprogramm für den monochromen Monitor. Bisher war es nur im ATARI-Entwicklungspaket erhältlich gewesen.
5. **Neochrom** – Für alle die es gern bunt mögen, ist dies ein Zeichenprogramm für den Farbmonitor. Vor allem die Demoprogramme sind besonders gelungen.
6. **Joshua** – Dies ist der Monitor und Diskmonitor, auf den schon kurz in diesem Heft eingegangen wurde. Fairerweise muß man zu Joshua sagen, daß dieses Programm von ATARI einfach auf ihre Diskette kopiert wurde, es aber auch schon vorher als Public-Domain-Programm zu haben war. Wer eine genaue Anleitung wünscht, muß sich direkt an den Autor wenden. Die Adresse steht in der Einschaltmeldung.

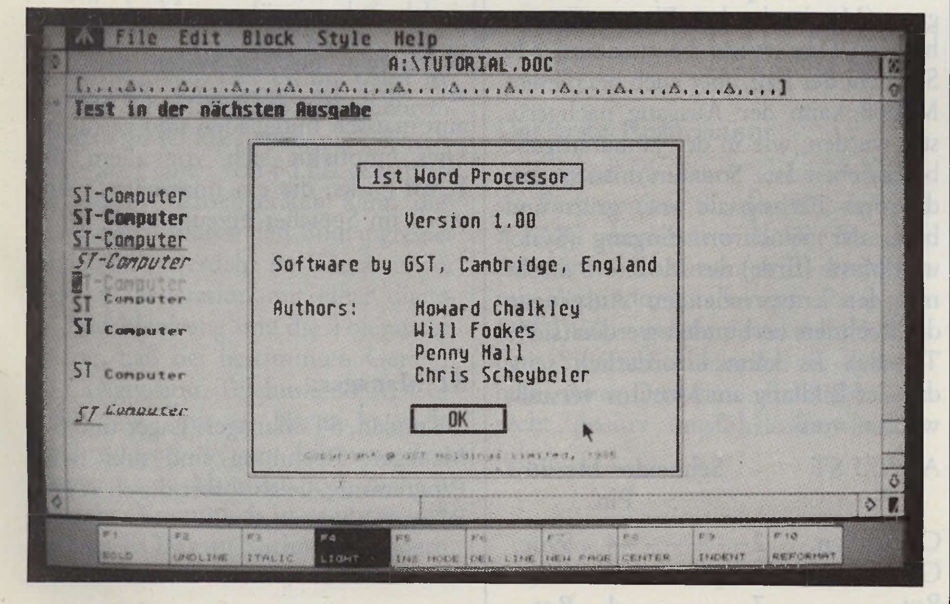


Bild 2

Alle diese Programme können, um es nochmal zu wiederholen, kostenlos bei jedem ATARI-Händler auf eigene

Disketten kopiert werden. Die oben beschriebenen DB Maser One, Doodle und GST-1st-Word sind von ATARI als Ersatz für die schon lange ausstehenden **GEM-Paint** und **GEM-Write** gedacht. Damit steht also fest, daß diese beiden Programme nicht mehr so schnell auf den deutschen Markt kommen werden.

Ferner gibt es noch einige Spiele als Public-Domain-Software, die allerdings nur bedingt oder gar nicht bei ATARI-Händler zu erhalten sind. Darunter befindet sich zum Beispiel **Megaroid** von der Firma Megamax.

Um beim Thema Spiele zu bleiben. Einige **Abenteuerspiele** (Adventures) sind inzwischen von anderen Rechnern für die ATARI ST-Serie angepaßt worden. Um nur eingie zu nennen : Zork I + II, Ultima II und Hitchhiker.

Nun zu einem weiteren wichtigen Punkt in diesen News. Es gibt eine **neue TOS**-Version. Sie hat 197744 Bytes und kann ebenfalls bei den Händlern kopiert werden. Es funktioniert jetzt endlich die serielle Schnittstelle (RS 232C) einschließlich dem XON/XOFF-Handshake-Signal einwandfrei. Außerdem stürzt das Betriebssystem nicht mehr nach einer größeren Anzahl von Diskettenzugriffen ab.

Wem das Textverarbeitungsprogramm der Firma GST gefällt, nachdem er es sich beim Händler kopiert hat, dem sei gesagt, daß es von der gleichen Firma ebenfalls einen unter GEM arbeitenden **C-Compiler** gibt, der auch in einer der nächsten Ausgaben von uns genauer unter die Lupe genommen wird.

Informationen

XY-Koordinaten

Will man im BASIC einen Kreis oder eine Linie an einen bestimmten Punkt im OUTPUT-Fenster erscheinen lassen, so muß man bei der Angabe die XY-Koordinaten wissen. Da die maximale Auflösung in X-Richtung 640 und in Y-Richtung 400 Pixel beträgt, könnte man annehmen, daß der Mittelpunkt des OUTPUT-Fensters bei der Hälfte der maximalen Pixel läge, also bei 320/200 Pixel. Dies ist jedoch nicht der Fall, da durch die Umrandung des Fensters einige Punkte nicht zu sehen sind. In X-Richtung hat man 616 und in Y-Richtung 345 Pixel zur Verfügung. In Y-Richtung ist also nur eine ungerade Anzahl von Bildpunkten vorhanden, so daß man zum Beispiel ein Koordinatensystem nie genau in der Mitte des Fensters darstellen kann. Die Koordinate 0,0 liegt in der oberen linken Ecke des OUTPUT-Fenster, entsprechend 615/344 in der rechten unteren Ecke.

Auch ohne Maus...

kann man den Mauszeiger auf dem Bildschirm bewegen und mit ihm alle Mausfunktionen ausführen. Die Cursorpfeil-Tasten zusammen mit der Alternate-Taste gedrückt, ergeben die jeweilige Richtung des Zeigers. Die linke Maus-Taste entspricht „Alternate + Insert" und die rechte „Alternate + Clr/Home". Die ganze Bewegung läßt sich noch durch zusätzliches Drücken der Shift-Taste verlangsamen. Bei manchen Operationen, wie zum Beispiel beim Kopieren von einzelnen Programmen, ist dies recht mühsam.

Auch die Escape-Taste hat einige zusätzliche Funktionen. So bekommt man, wenn man sich Inhaltsverzeichnisse von mehreren Disketten ansehen möchte, mit dieser Taste immer das aktuelle Verzeichnis, also das Verzeichnis der Diskette, die sich gerade im Laufwerk befindet. Man muß aber zuvor einmal ein Verzeichnis-Fenster geöffnet haben.

Schneider Farbmonitor am ATARI ST

Der RGB-Farbmonitor des CPC 464 kann problemlos an den ATARI ST angeschlossen werden, vorausgesetzt es ist der Composite Synchron-Ausgang (Monitorbuchse Pin zwei) vorhanden. Dies ist nur beim „alten" 520 ST nicht der Fall, aber auch bei diesem Modell kann der Ausgang nachgerüstet werden, wie in der Januarausgabe beschrieben ist. Sodann müssen nur die drei Farbsignale rot, grün und blau, der Synchron Eingang (Sync) und Masse (Erde) des Monitors direkt mit den entsprechenden Ausgängen des Rechners verbunden werden (siehe Tabelle). Es kann erforderlich sein, daß der Bildfang am Monitor verstellt werden muß.

ATARI ST	Pin		Schneider Monitor Pin	
Comp Syn.	2	——————	4	Sync.
Grün	6	——————	2	Grün
Rot	7	——————	1	Rot
Blau	10	——————	3	Blau
Masse	13	——————	5	Erde

Autostart

Es besteht beim Atari ST die Möglichkeit eine Autostartdatei zu erstellen. Eine solche Datei bewirkt, daß bei jedem Hard- und Softwarereset die Programme nachgeladen werden, die in der Autostartdatei stehen.

Um nun ein oder mehrere Programme automatisch zu starten, muß man folgende Anweisung befolgen.

1. Man legt auf der Systemdiskette einen neuen Ordner mit dem Namen **"AUTO"** an. Dabei wird keine Extension benötigt.
2. Jetzt kopiert man das bzw. die Programme, die automatisch gestartet werden sollen in diesen Ordner.

Die Programme müssen alle die Extension **"PRG"** haben. Gegebenenfalls muß man diese von ".TOS" oder "TTP" mit der Info-Option umbenennen.

Wenn man jetzt das System neu startet, wird das betreffende Programm automatisch mitgeladen und gestartet. Dies empfiehlt sich vor allem für RAM-Disks, die ein imaginäres Laufwerk im Speicher erzeugen.

ST Manager: ...

...Kunden, ST Manager: Lager und ST Manager: Rechnung sind drei neue Programme, die ab sofort durch ATARI vertrieben werden.

Erfahrungen beim Aufrüsten auf 1 Mega Byte

Zusätzlich zu den in der letzten Ausgabe angesprochenen Problemen nach dem Aufrüsten auf 1 Mega Byte, können Fehler durch folgende Tips eventuell beseitigt werden:

1. In verschiedenen Systemen liegt ein 47 pF Kondensator von Pin 39 des Videoshifters (U31) gegen Masse. Dieser Kondensator sollte entfernt werden. Er kann dann zwischen Pin 11 und Masse wieder eingesetzt werden.
2. Der Ausgang RAS1 (Pin 18) der MMU sollte in der Mitte des Silberdrahtes der zusätzlichen RAM-Bank angelötet werden, um so eine gleichmäßige Signalverteilung zu erreichen. Dasselbe gilt für die Einspeisung von RAS0 (Pin 8 der MMU).
3. In den Zuleitungen von RAS0 und RAS1 kann ein Widerstand von 68–100 Ohm zwischen der MMU und den RAMs eingesetzt werden. Die beiden Widerstände sollten möglichst nah an der MMU eingebaut werden.

Farbfernseher mit SCART-Buchse als Farbmonitor

In der letzten Ausgabe wurde diese Anschlußmöglichkeit beschrieben. Es ist jedoch hinzuzufügen, daß dabei nicht alle Farbfernsehgeräte verwendet werden können. Der Grund dafür ist, daß der ATARI ST ein Bildsynchronimpuls von 60 Hz liefert, die Fernsehgeräte aber einen von 50 Hz erwarten. Bei vielen Fernsehgeräten kann dies im Einzelfall durch den Bildfangregler nachgestellt werden. Bei Geräten der neueren Generation mit einer digitalen Bildablenkung sind die Toleranzen so eng, daß bei bestimmten Geräten (ITT Digivision, Telefunken PALCOLOR u. a.) mit 60 Hz nicht mehr synchronisiert werden können. Eine Lösung ist im Moment noch nicht in Aussicht. Vermutlich ist es notwendig die Boot-ROMs im Rechner zu ändern.

Fehlerkorrektur

Auf Seite 49 + 50 der Januarausgabe muß es in Zeilennummer 1030 und 1530 folgendermaßen heißen:

```
FOR A=NR TO 1 STEP -1.
```

Durch den Druck war das Minuszeichen bei „–1" leider schlecht zu lesen.

Bei dem Logo Programm-Beispiel SPIRAL auf Seite 60 hat sich ein weiterer Druckfehler eingeschlichen. Das richtige Listing lautet:

```
TO SPIRAL :SEITE :WINKEL :INS :INW
FD :SEITE
RT :WINKEL
SPIRAL :SEITE + :INS :WINKEL + :INW :INS : INW
END
```

Vorschau

Für den Amateurfunker:
Ein ausführlicher Testbericht über ein Funkfernschreibprogramm (RTTY).

Übersicht über die Befehle des 68 000er Prozessors.

Der 68 000er Prozessor unter die Lupe genommen.

Ausführliche Softwaretests:

- VIP Professional
- DB Master One
- 1st_Word

Fortsetzung der Kurse:

- GEM
- Pascal
- Logo

Der erste Farbmonitor

„Thomson for Atari" so heißt der erste Farbmonitor für den ST, der von ATARI vertrieben wird. Der Monitor der Firma Thomson wird komplett mit Verbindungskabel zum Preis von DM 1400,– geliefert und sollte bei jedem System-Fachhändler zu bekommen sein. Der erste Eindruck war recht positiv (ausführlicher Bericht folgt).

Kleinanzeigen

Ab nächster Ausgabe soll in der ST-Computer eine **Kleinanzeigenbörse** eröffnet werden.

Privatanzeigen kosten pro Zeile und Spalte DM 6,— incl. MwSt.

Gewerbliche Anzeigen kosten DM 7,— pro Zeile und Spalte.

Zur Einsendung bitte die beiliegende Karte verwenden.

Einsendeschluß ist der 7. Februar 1986.
Verspätete Einsendungen kommen in die April-Ausgabe.

Als erstes möchte ich Ihnen ein Lob über die Zeitschrift-Aufmachung und über den Inhalt machen.

Ich habe einen RGB-Monitor der Marke MICRO-VITEC Cub 653 mit einer RGB-TTL DIN Buchse. Den Monitor habe ich bisher mit einem Sinclair QL betrieben.

Meine Frage: Ist es möglich den Monitor an den ATARI 520ST anzuschließen? Ich hätte noch eine zweite Frage: Können Sie mir die Anschrift einer Firma mitteilen die eine Speichererweiterung macht?

Dietmar Fox
Garmisch

Antwort:

Wir bedanken uns für Ihren Brief.

a) Generell ist jeder RGB-Monitor problemlos an den ATARI anschließbar. Es ist jedoch bei einigen Modellen erforderlich den Bildfang anzugleichen, da der Rechner mit 60 Hz und der Monitor mit 50 Hz synchronisiert. Näheres finden Sie auf Seite 70 in dieser Ausgabe.

b) Es gibt grundsätzlich viele Firmen die diesen Umbau vornehmen. Eine der möglichen Adressen finden Sie unter den Inseraten dieser Ausgabe.

Die Redaktion

Seit kurzem bin ich stolzer Besitzer eines Atari 520 ST und habe beschlossen Ihre Zeitschrift regelmäßig zu kaufen, besonders, weil diese Zeitschrift verspricht, interessante, abzutippende Programme zu bringen.

Leider mußte ich in der ersten Ausgabe feststellen, daß das Grafikprogramm 'V1.0' auf den Seiten 48–51 nur bruchstückhaft läuft. Nach mühsamen Eintippen stand mir nur die Kuchengrafik vollständig zur Verfügung. Am Diagramm fehlte die Beschriftung und die Balkengrafik fehlte vollständig. Hiermit bitte ich Sie herzlich, mir die Lösung dieser Aufgabe zukommen zu lassen, bzw. in Ihrer nächsten Ausgabe auszudrucken. Für meine Lehrtätigkeit an einer Schule könnte ich das fertige Programm schnell gut gebrauchen.

Vielen Dank
Heinrich Will
Klingenberg

Antwort:

Es freut uns, daß Ihnen das Programm zusagt. Die bei Ihnen aufgetretenen Fehler lagen am etwas schwachen Druck. In den Zeilen 1030 und 1530 verschwanden bei einigen Heften die Minuszeichen. Dies hat zur Folge, daß diese zwei Schleifen nicht abgearbeitet werden, wodurch die Beschriftung (ab 1500) und das Balkendiagramm (ab 1000) nicht erscheint.

Ansonsten auftretende Fehler liegen am Abtippen, da das Listing nach abschließendem Test direkt ausgedruckt wurde.

An dieser Stelle wollen wir Sie auf unseren Diskettenservice hinweisen.

Die Redaktion

Rechnen mit ATARI ST

In meiner Not mit dem ST 520 wende ich mich an Sie und bitte um einen Tip. Im Dezember habe ich mir den Atari 520 ST zugelegt. Seit vielen Jahren arbeite ich mit einem Commodore 3032 und da dachte ich, daß ein bißchen technischer Fortschritt mir auch nutzen könne. Für viele Spielereien (auch ernsthafte) ist der Atari ST ja ein tolles Ding. Wehe aber, wenn der Rechner rechnen soll. Ich betreibe als Hobby Astronomie und da gibt es ja viele Dinge zu berechnen. Aber mit 7 Stellen Genauigkeit ist da nichts zu machen (da kann ein Millionär noch nicht einmal seine Pfennige zusammenkratzen!). Die Planeten werden aus ihrer Bahn ab- oder aufgerundet und sind nach wenigen Tagen nicht mehr zu finden. Auch finde ich es witzig, daß ohne eine Warnung alles was größer als 10 hoch 18 ist einfach zu Null wird.

War denn meine Erwartung zu hoch, als ich annahm, daß ein Computer Baujahr 1985 einem Taschenrechner wenigstens ebenbürtig sei?

Wird Atari denn ein gebrauchstüchtiges Basic jemals schaffen?

Vielleicht gelingt es Ihrer Zeitschrift der Firma etwas Dampf unter den Hintern zu machen oder aber Sie sollten fairerweise Kaufinteressenten, die auch noch Rechenambitionen haben vor dem Atari ST warnen.

Dr. Emil Jung
Marl

Antwort:

Ihre Enttäuschung wegen der Rechenungenauigkeit des mitgelieferten Basic-Interpreters können wir leider momentan nur teilen. Digital Research überarbeitet zur Zeit sämtliche Software, aber genaue Lieferungstermine sind uns unbekannt. Allerdings ist der ATARI ST kein reiner Basic-Rechner, deshalb ist es keineswegs berechtigt die sehr gute Hardware des Rechners wegen eines Softwarefehlers, der sicher bald behogen wird, zu verurteilen. Für höhere Ansprüche stehen diverse andere Sprachen wie z. B. C, Pascal oder Fortran zur Verfügung. Außerdem ist ein Basic-Modul von METACOMCO angekündigt.

Die ST-Redaktion